

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СТВОРЕННЯ ВІДГУКІВ

(назва теми)

ПРО САМОСТІЙНІ ПОДОРОЖІ АВТОМОБІЛЕМ. SOFTWARE

FOR REVIEWING INDEPENDENT CAR JOURNEYS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-110

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КОВАЛЬОВ Р.С.

(ПРІЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРІЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет _____ КНТ _____
Кафедра _____ програмних засобів _____
Ступінь вищої освіти _____ бакалавр _____
Спеціальність _____ 121 Інженерія програмного забезпечення _____
(код і найменування)
Освітня програма (спеціалізація) _____ Інженерія програмного забезпечення _____
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф. _____
Сергій СУББОТІН
« _____ » _____ 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КОВАЛЬОВА Ростислава Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення створення відгуків про самостійні подорожі автомобілем. Software for Reviewing Independent Car Journeys

керівник проєкту (роботи) к.т.н., доцент СКРУПСЬКИЙ Степан Юрійович _____,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 24 » квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Реалізація програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 15 » квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

(підпис) Ростислав КОВАЛЬОВ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Степан СКРУПСЬКИЙ
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 88 с., 21 рис., 2 табл., 3 дод., 8 джерел.

АВТОМОБІЛЬ, ПОДОРОЖ, СИСТЕМА ВІДГУКІВ, МАРШРУТ, ВЕБДОДАТОК.

Об'єкт роботи – процес самостійного подорожування автомобілем.

Предмет роботи – програмне забезпечення створення відгуків про самостійні подорожі автомобілем.

Мета роботи – розробити програмне забезпечення створення відгуків про самостійні подорожі автомобілем для обміну інформацією стосовно можливостей відвідування різних місць автомобілем.

Матеріали, методи та технічні засоби: мова Python, система MySQL, фреймворк Django.

Результати. Створено програму, щоб підтримувати самостійні подорожі автомобілем, дозволяючи надавати інформацію про досвід інших користувачів, шукати нові варіанти для подорожей, ділитися власним досвідом.

Висновки. У роботі вдалось виконати аналіз аналогів, який призвів до виділення можливостей, які доцільно в першу чергу реалізувати в програмі, забезпечити проєктування, що стало підґрунтям для реалізації програми, де вдалось забезпечити як опис досвіду подорожі користувачем, так і створення маршрутів подорожей як через визначення окремих точок на мапі, так і через визначення широковідомих доріг, що сприяє подальшому пошуку такої інформації подорожуючими. Тестування програми створення відгуків про самостійні подорожі автомобілем проведено успішно.

Галузь використання. Подорожі автомобілем індивідуально або групами.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 88 pages, 21 figures, 2 tables, 3 appendixes, 8 sources.

CAR, TRAVEL, REVIEWING SYSTEM, ROUTE, WEB APPLICATION.

The object of the work is the process of traveling independently by car.

The subject of the work is the software for creating reviews about independent car trips.

The purpose of the work is to develop software for creating reviews about independent car trips to exchange information about the possibilities of visiting different places by car.

Materials, methods and technical means: Python language, MySQL system, Django framework.

The results. The program was created to support independent car travels, allowing to provide information about other users' experiences, search for new travel options and share own experience.

Conclusions. In the work, an analysis of analogues was carried out. It led to the selection of opportunities that should first of all be implemented in the program. Then software design was provided. It became the basis for the implementation of the program, where it was possible to provide both a description of the user's travel experience and the creation of travel routes both through the definition of individual points on the map, as well as through the definition of well-known roads, which facilitates the further search of such information by travelers. The testing of the program for creating reviews about independent car trips was successfully completed.

Field of use. Travel by car individually or in groups.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	9
1.1 Аналіз додатків для самостійних подорожей автомобілем	9
1.2 Порівняльний аналіз розробки та аналогів	12
1.3 Висновки за розділом 1	13
2 Розробка архітектури програми.....	14
2.1 Вибір засобів проєктування	14
2.2 Сценарії використання програми	15
2.3 Структура бази даних	18
2.4 Висновки за розділом 2	29
3 Реалізація програми	30
3.1 Визначення порядку роботи з програмою.....	30
3.2 Результати реалізації програмних одиниць додатку	31
3.3 Висновки за розділом 3	49
4 Експлуатація та тестування програми	50
4.1 Призначення програми	50
4.2 Умови виконання програми	50
4.3 Виконання і тестування програми.....	50
4.4 Висновки за розділом 4	55
Висновки	56
Перелік джерел посилання	57
Додаток А Технічне завдання	58
Додаток Б Текст програми	63
Додаток В Слайди презентації.....	82

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

MVT – Model-View-Template;

БД – база даних.

ВСТУП

Подорожі можуть відбуватися в різних варіантах. Одним з таких варіантів є самостійні подорожі автомобілем. Особливістю цього типу подорожей є те, що самі подорожуючі формують маршрут подорожі, а тому досвід інших людей, які вже відвідували ці місця автомобілем, є корисним. Тому потрібно створити програму, яка дозволить для самостійних подорожей визначати маршрути, ділитися інформацією про виконані подорожі на основі цих маршрутів, залишати відгуки про використані для таких подорожей автомобілем маршрути та зручно шукати потрібну інформацію серед всієї відповідної накопиченої.

При цьому важливо підкреслити, що відгуки розуміються тут в доволі широкому сенсі:

- з одного боку це відгуки про маршрути або дороги, тобто те, що вже визначено в програмі іншими користувачами (в тому числі адміністратором), але потребує накопичення думок для того, щоб ті користувачі, які збираються використати цей маршрут або проїхати цією дорогою, розуміли, на скільки це варто того;

- з іншого боку це відгуки про власний досвід подорожей як таких, тобто коли це фактично опис власної подорожі, але при цьому в такому описі визначається маршрут подорожі.

Важливо, щоб в програмі відгуки розглядалися саме з обох боків. Це має сприяти тому, щоб користувачі ділилися власним досвідом як у форматі подорожей, так і форматі відвіданих доріг, маршрутів, що за рахунок зв'язування подорожей з маршрутами, а тих у свою чергу з дорогами має сприяти розширенню наявної інформації у програмі та позитивно відбиватися на підтримці самостійних подорожей автомобілем.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз додатків для самостійних подорожей автомобілем

Подорожі автомобілем є досить популярним варіантом, але підтримка цього процесу є не на стільки розповсюдженою особливо стосовно створення відгуків про такі подорожі. Оскільки, як було зазначено у вступі, відгуки розуміються в достатньо широкому сенсі, то потрібно, щоб і в програмі вони були відповідним чином підтримуваними.

Furkot [1] (рис. 1.1) є програмним інструментом, який придатний для планування подорожей. Тобто він фактично дозволяє визначити маршрут, за яким можна переміщатися в основі цієї подорожі, готуючи її індивідуально. Це важливо при плануванні нових маршрутів і є зручним за рахунок виділення автомобільних шляхів на мапах та прокладання таким чином маршруту.

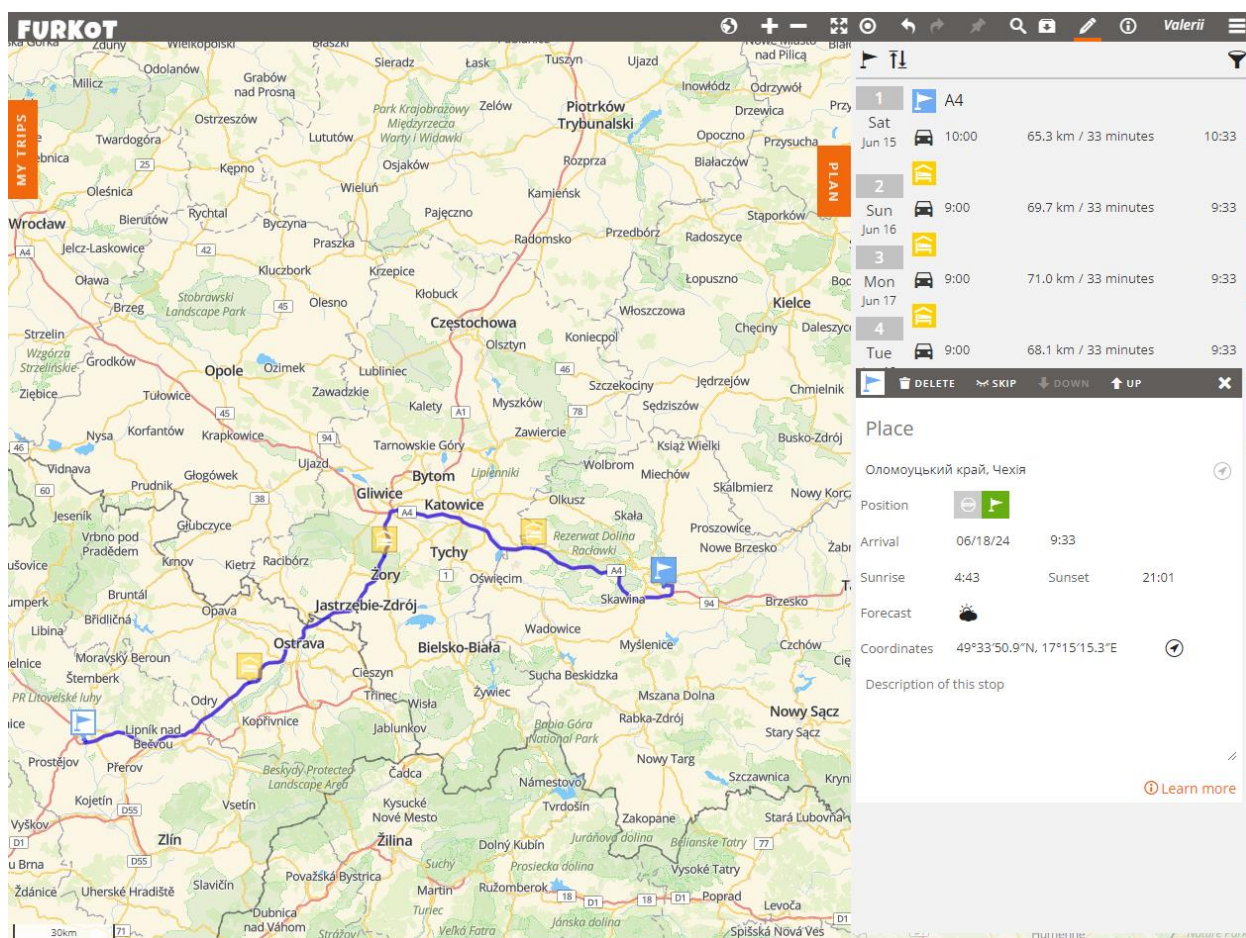


Рисунок 1.1 – Додаток Furkot

Таким чином можна одразу мати інформацію про те, скільки така подорож займає часу тощо.

Іншим типом додатків, які можуть бути задіяні стосовно самостійних подорожей автомобілем, є блоги, де люди описують свій власний досвід. Прикладом таких блогів зокрема є Lifejourney4two [2] (рис. 1.2). Цей блог створений за принципом надання керівництв для подорожей.



Рисунок 1.2 – Додаток Lifejourney4two

Відповідно персонального, можливо емоційного досвіду окремих подорожей в цьому блозі немає. Такі керівництва можуть бути корисними, однак не є описом вражень щонайменше. Окрім того цей блог не дозволяє шукати інформацію на основі маршрутів. Доступ фактично можна виконати за розділенням за країнами. Самі керівництва містять описи маршрутів у тому числі на мапі з пунктами і визначеним маршрутом. Однак для доступу до маршруту потрібно дійти до потрібного керівництва. Пошук відбувається за загальним текстом, тому відділити певні характеристики маршруту або сам маршрут у такому пошуку неможливо.

Блог Олени та Євгенія Пузатків [3] (рис. 1.3) є прикладом персонального блогу, де фактично як раз представлені персональні враження про окремі поїздки. Недоліком є зокрема те, що маршрути демонструються як зображення загального маршруту. Тобто деталізувати на мапі, перенести на мапу такий маршрут стає досить складно і потрібно виконувати для цього додаткову роботу. Також за блогом організований звичайний текстовий пошук. Звичайно, що цінність такого пошуку не надто би підвищилась з додаванням інших параметрів, однак цим і виявляється ще одна проблема блогів – зазвичай це окремі персональні вебсайти, а тому охопити інформацію від різних користувачів неможливо. Тоді треба шукати такі окремі блоги, знаходити з них потрібну інформацію, що збільшує час пошуку.

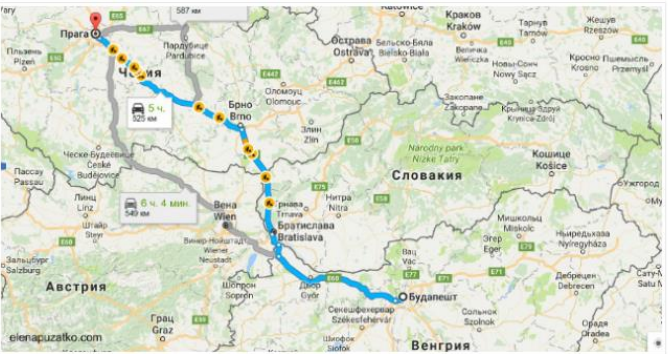
Головна	Про нас	Країни	Поради мандрівникам	Бронюємо самі	Послуги	Путівники	Контакти
= <u>Звіт про подорож – Частина 1. Україна, Сколе, замок Паланок, Берегове</u> Після пробігу по Україні у нас було пару днів відпочинку і акліматизації до європейського ритму життя. Провели ми ці дні в гостях у наших рідних в передмісті Будапешта – Будакесі. Читайте наші статті про Угорщину тут: = <u>Подорожі Угорщиною</u> = <u>На машині в Угорщину. Оплата доріг, ПДР, комплектація авто, вартість бензину, місця для відпочинку в дорозі.</u> І ось настав довгоочікуваний момент. Головний етап нашої подорожі – пробіг по таким європейським країнам: Словаччина, Чехія і Німеччина. Першим пунктом поїздки була Прага. За день запланували проїхати 525 км від Будапешта до Праги. Наш маршрут на мапі:  Хвилювання додавав той факт, що нам потрібно було за один день проїхати два кордони і три країни. У нас в подорожах було подолання трьох кордонів, наприклад, На машині до Болгарії. Але тут інша ситуація, цікаво побачити відсутність офіційних кордонів всередині Євросоюзу. Частина маршруту проходила по території Словаччини. Ми ще в Україні через інтернет сплатили словацькі дороги. Як це зробити читайте в цій статті: = <u>На машині до Словаччини. Оплата доріг, ПДР, комплектація авто, вартість бензину</u> Звичайно, є в цій країні варіант проїзду безкоштовними дорогами, з маленькою втратою в часі. Але ми вибрали платні автобани, адже у нас головна мета на цьому етапі не Словаччина, а саме Прага. Угорськими прекрасними дорогами наближаємося до Словаччини.					Натиснувши кнопку, ви погоджуєтесь з політикою конфіденційності та обробки персональних даних. Пошук на блозі Пошук у блозі ...  Бронювання готелів Booking.com Ми користуємось сайтами: Booking.com - наш улюблений сервіс пошуку готелів і апартаментів. Skyscanner.com.ua - пошук дешевих авіаквитків Infobus - міжнародний сервіс пошуку квитків на автобуси та потяги по Україні та Європі. Rentalcars.com - оренда автомобіля в будь-якій країні світу. - онлайн-страхування: Туристична страховка: знижка 10%. Промокод EP19 (літери англійською). Зелена картка: Бонус 7%. Промокод EPGC7.		

Рисунок 1.3 – Блог Олени та Євгенія Пузатків

Прикладом вебсайту, де відбувається збирання інформації від різних авторів, є Driving [4]. Акцентом даного вебсайту є те, що тут описується не тільки досвід подорожі, але це робиться в ракурсі конкретного автомобіля, на якому ця подорож відбувалась, що звичайно може бути додатковою корисною інформацією. Однак тут не представлені конкретні маршрути таких подорожей на мапі, відсутній спеціалізований пошук.

1.2 Порівняльний аналіз розробки та аналогів

Остаточний порівняльний аналіз розробки програми створення відгуків про самостійні подорожі автомобілем та аналогів містить Furkot та об'єднані в єдину групу блоги (табл. 1.1).

Таблиця 1.1 – Порівняння розробки програми створення відгуків про самостійні подорожі автомобілем та аналогів

Критерій	Furkot	Блоги	Розробка
Пошук маршрутів	Відсутній	Тільки загальний текстовий	+
Створення власних маршрутів користувачем	+	—	+
Визначення власного досвіду подорожей	—	+	+
Створення відгуків про наявні маршрути	—	—	+

Розробка програми створення відгуків про самостійні подорожі автомобілем в цій роботі повинна мати пошук маршрутів не тільки за текстом (назвою), але і за додатковими параметрами, дозволяти створювати відгуки

про наявні маршрути, що є відмінностями цієї програми порівняно з проаналізованими.

1.3 Висновки за розділом 1

Аналіз програм для підтримки самостійних подорожей автомобілем показав, що вони розділяються на програми-планувальники маршрутів подорожей та персональні блоги. Проте важливим є об'єднання цих можливостей, щоб інші користувачі могли не тільки використовувати, але і створювати відгуки про такий досвід використання маршрутів інших користувачів, при цьому ділитися власним досвідом.

Завдання на роботу:

- проаналізувати програми створення відгуків про самостійні подорожі автомобілем;
- визначити засоби проєктування програми;
- виділити вимоги до програми створення відгуків про самостійні подорожі автомобілем;
- виконати реалізацію програми створення відгуків про самостійні подорожі автомобілем.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів проєктування

Для розробки вебдодатку на рівні, який відповідає сучасному, було розглянуто фреймворки для веброзробки, включаючи Django [5] як найвідоміший Python-вебфреймворк та CherryPy [6]-[7] як один з найновіших вебфреймворків для Python (табл. 2.1).

Таблиця 2.1 – Вибір фреймворку для розробки програми створення відгуків про самостійні подорожі автомобілем

Критерій	CherryPy	Django
Тип	Мініфреймворк	Мегафреймворк
Продуктивність	Висока	Прийнятна для використання в реальних додатках
Забезпечення підтримки баз даних	Потрібно обрати, встановити і використати фреймворк об'єктно-реляційного відображення	Усі необхідні засоби початково вбудовані
Адміністративна панель	Як такої вбудованої немає	Має вебінтерфейс, дозволяє змінювати, додавати, вилучати дані всіх моделей
Рушій для роботи з шаблонами	Потрібно обрати, встановити і використати	Вбудований власний (може бути змінений)
Безпека	Вбудованих інструментів немає	Великий набір вбудованих інструментів

Основна відмінність полягає в тому, що Django – мегафреймворк (має багато вбудованих засобів), а CherryPy – мініфреймворк (має тільки основу). За рахунок цього CherryPy забезпечує кращу продуктивність, хоча результати Django прийнятні для використання в реальних умовах, в тому числі за великих навантажень. З іншого боку Django має всі необхідні засоби для зменшення витрат на розробку, пришвидшення її, спрощення, що стосується як роботи з базами даних, так і з шаблонами вебсторінок, безпекою даних. Підсумок – вибір Django для розробки.

2.2 Сценарії використання програми

Визначаючи сценарії використання програми, необхідно було визначити акторів для них. Акторами загалом є звичайний користувач і адміністратор. Робота адміністратора забезпечується адміністративною панеллю Django з доступом до всіх внесених даних, тому сценарії для адміністратора окремо не виділялись. Основний потік сценаріїв користувача програми створення відгуків про самотійні подорожі автомобілем:

- формування маршруту, який може використовуватися для самотійної подорожі автомобілем;
- перегляд запису широковідомої дороги;
- перегляд всіх відгуків за широковідомою дорогою;
- перегляд всіх подорожей за широковідомою дорогою;
- перегляд маршруту самотійної подорожі автомобілем;
- перегляд всіх відгуків за маршрутом самотійної подорожі автомобілем;
- перегляд всіх подорожей за маршрутом самотійної подорожі автомобілем;
- авторизація;
- додавання інформації про використовуваний автомобіль користувачем для самотійних подорожей;

- перегляд даних користувача;
- опис виконаної самотійної подорожі автомобілем;
- перегляд виконаної самотійної подорожі автомобілем;
- перегляд всіх самотійних подорожей автомобілем користувача;
- створення відгуку про маршрут самотійної подорожі;
- створення відгуку про широковідому дорогу;
- збереження широковідомої дороги;
- збереження маршруту самотійної подорожі;
- перегляд збережених доріг;
- перегляд збережених маршрутів самотійних подорожей;
- визначення відвідування дороги;
- перегляд всіх відвіданих користувачем доріг;
- перегляд відгуків користувача про маршрути самотійної подорожі;
- перегляд відгуків користувача про дороги;
- пошук маршрутів самотійних подорожей автомобілем;
- пошук виконаних самотійних подорожей автомобілем;
- пошук населених пунктів;
- перегляд запису населеного пункту.

Але це основні сценарії, а до них є певні альтернативні, які виконуються за певних умов:

- вилучення збереженої широковідомої дороги є альтернативою до збереження широковідомої дороги та задіюється при існуючому збереженні, яке виконано помилково або не є більше потрібним;

- вилучення збереженого маршруту самотійної подорожі є альтернативою до збереження маршруту самотійної подорожі та задіюється при існуючому збереженні, яке виконано помилково або не є більше потрібним;

- реєстрація є альтернативою до авторизації, коли відсутній обліковий запис у користувача (рис. 2.1).

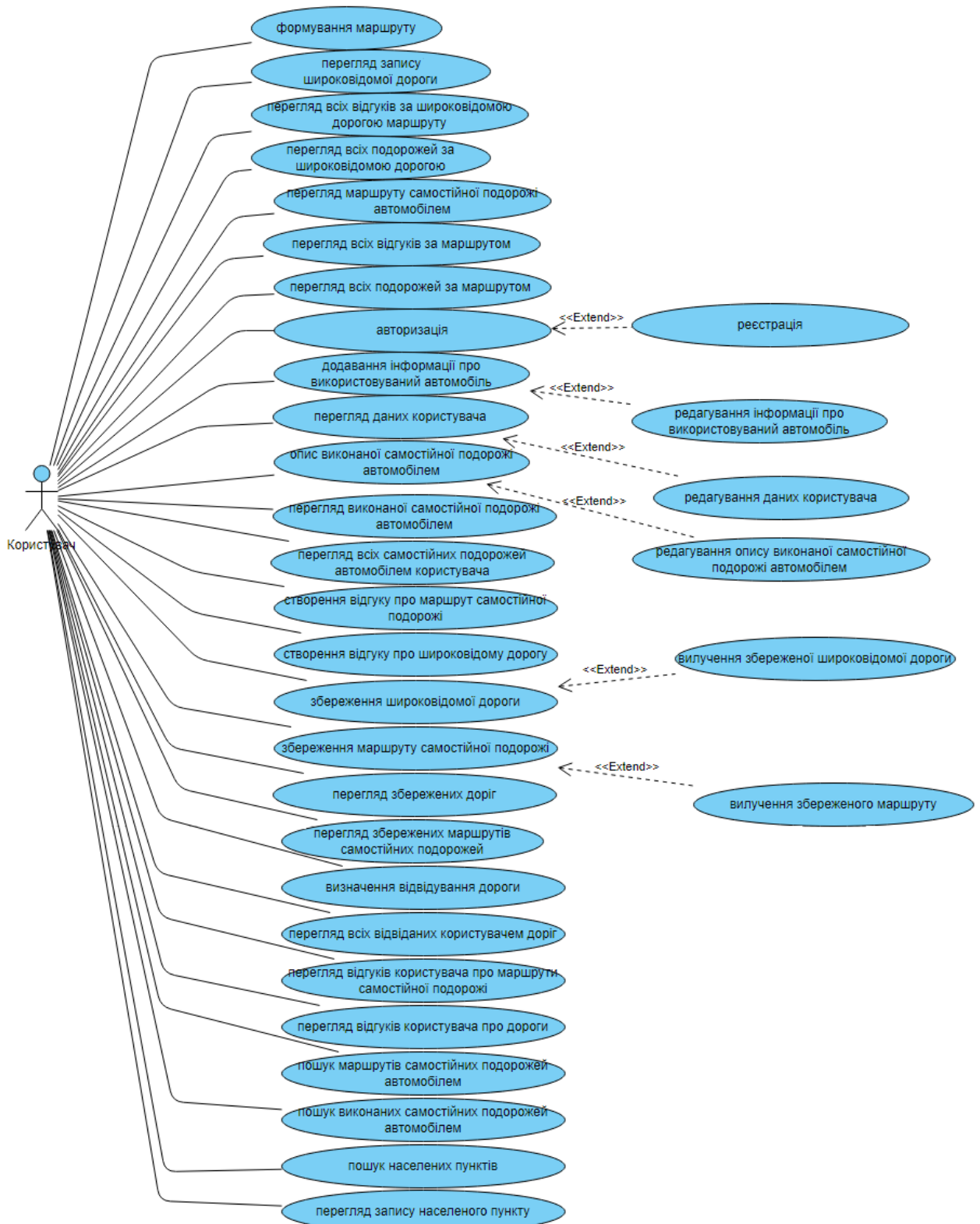


Рисунок 2.1 – Діаграма прецедентів програми створення відгуків про самостійні подорожі автомобілем

Альтернативними сценаріями також є:

– редагування інформації про використовуваний автомобіль користувачем для самостійних подорожей є альтернативою до додавання інформації про використовуваний автомобіль користувачем для самостійних подорожей та задіюється при внесеному записі про такий автомобіль, що належить користувачу, який розпочинає редагування;

– редагування опису виконаної самостійної подорожі автомобілем є альтернативою до опису виконаної самостійної подорожі автомобілем та задіюється при внесеному описі, що належить користувачу, який розпочинає редагування;

– редагування даних користувача є альтернативою до перегляду даних користувача, якщо дані не відповідають дійсності та належать користувачу, який розпочинає редагування.

2.3 Структура бази даних

База даних (БД), що створена в роботі, є реляційною, тому структура визначається на основі таблиць та зв'язків, які між цими таблицями існують. Саме це і описано далі, враховуючи при цьому, що керування БД відбувається через MySQL, що має переваги забезпечення великою кількістю можливостей, що може бути важливим при масштабуванні системи, окрім того є безкоштовним.

Таблиця Driveposition містить у БД інформацію про місце на мапі. Слід окремо вказати, що місце на мапі необхідне для того, щоб сформувати маршрут поїздки. Зважаючи на те, що це поїздки автомобілем, то такий маршрут може включати або цілі дороги, або окремі точки, а ці точки в свою чергу знаходитися в певних населених пунктах або поза межами них, але в підсумку в сукупності вони дозволяють сформувати маршрут подорожі. Місця на мапі в таблиці Driveposition визначені полями:

– id – номер, що унікально ідентифікує місце на мапі;

- position_name – назва місця на мапі (може бути не вказаною, тому допустиме порожнє значення);
- position_feature – коментар до особливості цього місця (може бути не вказаною, тому допустиме порожнє значення);
- position_longitude – довгота точки на мапі цього місця;
- position_latitude – широта точки на мапі цього місця;
- drivesettlement – населений пункт, в межах якого знаходиться це місце на мапі (може знаходитись поза межами населеного пункту, тоді допустиме порожнє значення).

Таблиця Drivesettlement містить у БД інформацію про населені пункти, визначену полями:

- id – номер, що унікально ідентифікує населений пункт;
- drivecountry – країна, в якій знаходиться населений пункт (це зв'язуюче поле, що належить до таблиці Drivecountry);
- settlement_name – назва населеного пункту.

Таблиця Drivecountry містить інформацію про країни, з якими працює програма і які можуть використовуватися в маршрутах самостійних подорожей автомобілем, визначену полями:

- id – номер, що унікально ідентифікує країну, що може використовуватися в маршрутах самостійних подорожей автомобілем;
- country_name – назва країни, що може використовуватися в маршрутах самостійних подорожей автомобілем.

Маршрут, який може використовуватися для самостійної подорожі автомобілем, задається таблицею Drivetrack, визначеною полями:

- id – номер, що унікально ідентифікує маршрут, який може використовуватися для самостійної подорожі автомобілем;
- driveruser – водій, який є зареєстрованим користувачем та який створив цей маршрут у програмі (це зв'язуюче поле, що належить до таблиці Driveruser);

– `track_name` – назва маршруту, який може використовуватися для самостійної подорожі автомобілем;

– `track_review` – оцінка, яку на даний момент усереднено отримав цей маршрут, який може використовуватися для самостійної подорожі автомобілем.

Такі маршрути можуть або формуватися під конкретну подорож автомобілем користувача, або далі типово використовуватися для подорожей інших користувачів або можливо для повторних подорожей того самого користувача. Тому доцільним було винести сам маршрут поза межі сутності подорожі.

Маршрути можуть складатися або з конкретних місць на мапі, які описувалися вище, або з проїзду певною дорогою, яку зазвичай можна вважати або широковідомою, або цікавою для самостійних подорожей автомобілем. Окрім того маршрут може охоплювати і окремі місця на мапі, і маршрути. Конкретні місця на мапі, які входять до складу маршруту самостійної подорожі автомобілем, задаються таблицею `Drivetrackposition`, визначеною полями:

– `id` – номер, що унікально ідентифікує конкретне місце на мапі, яке входить до складу маршруту самостійної подорожі автомобілем;

– `drivetrack` – маршрут, який може використовуватися для самостійної подорожі автомобілем і до складу якого входить це місце на мапі (це зв'язуюче поле, що належить до таблиці `Drivetrack`);

– `driveposition` – місце на мапі, яке входить до маршруту, який може використовуватися для самостійної подорожі автомобілем (це зв'язуюче поле, що належить до таблиці `Driveposition`);

– `position_feature` – особливості цього місця на мапі у складі даного маршруту, який може використовуватися для самостійної подорожі автомобілем (це поворот, рекомендована зупинка, цікаве місце для відвідання або щось інше): може бути не вказаною, тому допустиме порожнє значення;

– journey_number – позиція цього місця на мапі в порядку маршруту, який може використовуватися для самостійної подорожі автомобілем: це позиція не тільки серед всіх місць на мапі у складі маршруту, але і серед доріг, які також можуть бути додані до маршруту.

Друга частина складу маршруту, який може використовуватися для самостійної подорожі автомобілем, як було зазначено, – широковідомі дороги. Входження дороги до складу маршруту задається таблицею Drivetrackroad, визначеною полями:

– id – номер, що унікально ідентифікує входження дороги до складу маршруту, який може використовуватися для самостійної подорожі автомобілем;

– drivetrack – маршрут, який може використовуватися для самостійної подорожі автомобілем і до складу якого входить ця широковідома дорога (це зв’язуюче поле, що належить до таблиці Drivetrack);

– road – широковідома дорога, яка входить до маршруту, який може використовуватися для самостійної подорожі автомобілем (це зв’язуюче поле, що належить до таблиці Road);

– journey_number – позиція цієї дороги в порядку маршруту, який може використовуватися для самостійної подорожі автомобілем: це позиція не тільки серед всіх доріг, але і серед місць на мапі у складі маршруту, які також можуть бути додані до маршруту.

Широковідомі дороги, по яким здійснюють самостійні подорожі автомобілем, задаються таблицею Road, визначеною полями:

– id – номер, що унікально ідентифікує широковідому дорогу, якою здійснюють самостійні подорожі автомобілем;

– road_name – назва широковідомі дороги, якою здійснюють самостійні подорожі автомобілем;

– road_review – оцінка, яку на даний момент усереднено отримала ця широковідома дорога.

Фотографії широковідомих доріг задаються таблицею Roadphoto, визначеною полями:

- id – номер, що унікально ідентифікує фотографію широковідомої дороги;
- road – широковідома дорога, подорож якою зображена на фотографії (це зв'язуюче поле, що належить до таблиці Road);
- photo – сама фотографія широковідомої дороги.

Місце на мапі, що визначає позицію широковідомої дороги, задається таблицею Roadposition, визначеною полями:

- id – номер, що унікально ідентифікує місце на мапі, що визначає позицію широковідомої дороги;
- road – широковідома дорога, яка включає це місце на мапі (це зв'язуюче поле, що належить до таблиці Road);
- position_feature – особливості цього місця на мапі стосовно даної широковідомої дороги (може бути не вказаною, тому допустиме порожнє значення);
- position_longitude – довгота точки на мапі цього місця з широковідомої дороги;
- position_latitude – широта точки на мапі цього місця з широковідомої дороги;
- drivecountry – країна, де це місце знаходиться (це зв'язуюче поле, що належить до таблиці Drivecountry).

Зареєстровані користувачі задаються таблицею Driveruser, визначеною полями:

- id – номер, що унікально ідентифікує зареєстрованого користувача, що може здійснювати самостійні подорожі автомобілем;
- auth_user – обліковий запис з Django, який використовується зареєстрованим користувачем (це зв'язуюче поле, що належить до таблиці User з Django);

– driver_name – ім'я водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем;

– birth_date – дата народження водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем;

– avatar – фотографія водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем;

– driver_experience – досвід водіння водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем;

– driver_about – інформація про водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем;

– drivesettlement – населений пункт, в якому проживає водій, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем (це зв'язуюче поле, що належить до таблиці Drivesettlement).

Автомобіль водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем, задається таблицею Drivercar, визначеною полями:

– id – номер, що унікально ідентифікує автомобіль водія, який є зареєстрованим користувачем;

– driveruser – зареєстрований користувач, який є водієм цього автомобіля (це зв'язуюче поле, що належить до таблиці Driveruser);

– carmanufmodel – модель автомобіля, яким користується водій, який є зареєстрованим користувачем, для самостійних подорожей автомобілем (це зв'язуюче поле, що належить до таблиці Carmanufacturer);

– `produced_year` – рік виробництва автомобіля, яким користується водій, який є зареєстрованим користувачем, для самостійних подорожей автомобілем;

– `transmission` – тип коробки передач, який використовується в автомобілі (автоматична, механічна), що користується водієм, який є зареєстрованим користувачем, для самостійних подорожей автомобілем;

– `engine` – об'єм двигуна автомобіля, яким користується водій, який є зареєстрованим користувачем, для самостійних подорожей автомобілем;

– `car_photo` – фотографія автомобіля, яким користується водій, який є зареєстрованим користувачем, для самостійних подорожей автомобілем;

– `car_about` – опис автомобіля, яким користується водій, який є зареєстрованим користувачем, для самостійних подорожей автомобілем;

– `owner_since` – рік, з якого є власником цього автомобіля (або фактично використовує його) цей зареєстрований користувач.

Виробники автомобілів, які доступні в програмі, задаються таблицею `Carmanufacturer`, визначеною полями:

– `id` – номер, що унікально ідентифікує виробника автомобіля, доступного в програмі;

– `manuf_name` – назва виробника автомобіля, доступного в програмі.

Моделі виробників автомобілів, які доступні в програмі, таблицею `Carmanufmodel`, визначеною полями:

– `id` – номер, що унікально ідентифікує модель автомобіля, доступну в програмі;

– `carmanufacturer` – виробник, який виготовляє цю модель автомобіля, доступну в програмі (це зв'язуюче поле, що належить до таблиці `Carmanufacturer`);

– `model_name` – назва моделі автомобіля, яка доступна в програмі.

Самостійна подорож автомобілем, яку вже здійснив зареєстрований користувач, задається таблицею БД `Carjourney`, визначеною полями:

– id – номер, що унікально ідентифікує самотійну подорож автомобілем, яку вже здійснив зареєстрований користувач;

– drivercar – автомобіль водія, який є зареєстрованим користувачем, яким було здійснено цю самотійну подорож автомобілем (це зв'язуюче поле, що належить до таблиці Drivercar);

– journey_name – назва самотійної подорожі автомобілем, яку було здійснено;

– journey_about – опис самотійної подорожі автомобілем, яку було здійснено;

– journey_start – дата початку самотійної подорожі автомобілем, яку було здійснено;

– journey_end – дата закінчення самотійної подорожі автомобілем, яку було здійснено;

– total_kilometers – загальна довжина самотійної подорожі автомобілем, яку було здійснено, в кілометрах;

– total_capacity – загальна витрата палива на самотійну подорож автомобілем, яку було здійснено.

Одна самотійна подорож автомобілем, яку здійснив зареєстрований користувач, може охоплювати декілька маршрутів, які були раніше задані цим або іншими користувачами, або включати тільки один маршрут. Те, які маршрути входять до самотійної подорожі автомобілем, задається таблицею Carjourneytrack, визначеною полями:

– id – номер, що унікально ідентифікує входження маршруту самотійної подорожі автомобілем до конкретної подорожі зареєстрованим користувачем;

– carjourney – самотійна подорож автомобілем, яку було здійснено зареєстрованим користувачем з використанням цього маршруту (це зв'язуюче поле, що належить до таблиці Carjourney);

– drivetrack – маршрут самостійної подорожі автомобілем, який було використано в цій подорожі (це зв'язуюче поле, що належить до таблиці Drivetrack).

Фотографії самостійної подорожі автомобілем, яку було здійснено, задаються таблицею Carjourneyphoto, визначеною полями:

– id – номер, що унікально ідентифікує фотографію самостійної подорожі автомобілем;

– carjourney – самостійна подорож автомобілем, до якої належить ця фотографія (це зв'язуюче поле, що належить до таблиці Carjourney);

– photo – сама фотографія з самостійної подорожі автомобілем.

Відгуки на маршрути самостійних подорожей автомобілем задаються таблицею Drivetrackreview, визначеною полями:

– id – номер, що унікально ідентифікує відгук на маршрут самостійної подорожі автомобілем;

– drivetrack – маршрут самостійної подорожі автомобілем, на який створений цей відгук (це зв'язуюче поле, що належить до таблиці Drivetrack);

– driveruser – зареєстрований користувач, що створив цей відгук на маршрут самостійної подорожі автомобілем (це зв'язуюче поле, що належить до таблиці Driveruser);

– reviewed_time – момент створення відгуку на маршрут самостійної подорожі автомобілем;

– reviewed_text – текст відгуку на маршрут самостійної подорожі автомобілем;

– reviewed_point – оцінка з відгуку на маршрут самостійної подорожі автомобілем.

Збереження маршрутів, які можуть використовуватися для самостійної подорожі автомобілем, користувачем задаються таблицею Drivetracksave, визначеною полями:

– id – номер, що унікально ідентифікує збереження маршруту, який може використовуватися для самостійної подорожі автомобілем, користувачем;

– drivetrack – маршрут самостійної подорожі автомобілем, який було збережено користувачем (це зв'язуюче поле, що належить до таблиці Drivetrack);

– driveruser – зареєстрований користувач, який зберіг цей маршрут, який може використовуватися для самостійної подорожі автомобілем (це зв'язуюче поле, що належить до таблиці Driveruser).

Відгуки на самостійні подорожі широковідомими дорогами задаються таблицею Roadreview, визначеною полями:

– id – номер, що унікально ідентифікує відгук н самостійну подорож широковідомою дорогою;

– road – широковідома дорога, на самостійну подорож якою створив відгук користувач (це зв'язуюче поле, що належить до таблиці Road);

– driveruser – зареєстрований користувач, який подорожував самостійно автомобілем цією широковідомою дорогою і створив цей відгук (це зв'язуюче поле, що належить до таблиці Driveruser);

– visit_date – дата, коли цей зареєстрований користувач подорожував самостійно автомобілем цією широковідомою дорогою;

– reviewed_time – дата, коли було створено цей відгук на самостійну подорож широковідомою дорогою;

– reviewed_text – текст відгуку на самостійну подорож широковідомою дорогою;

– reviewed_point – оцінка відгуку на самостійну подорож широковідомою дорогою.

Відвідування широковідомої дороги може визначатися в програмі і без створення відгуку, щоб тільки накопичити собі історію власних подорожей. Такі дані задаються таблицею Roadvisit, визначеною полями:

- id – номер, що унікально ідентифікує відвідування широковідомої дороги користувачем;
 - road – широковідома дорога, яку було відвідано користувачем (це зв'язуюче поле, що належить до таблиці Road);
 - driveruser – користувач, який відвідав цю широковідому дорогу (це зв'язуюче поле, що належить до таблиці Driveruser);
 - visit_date – дата відвідування широковідомої дороги користувачем
- (рис. 2.2).

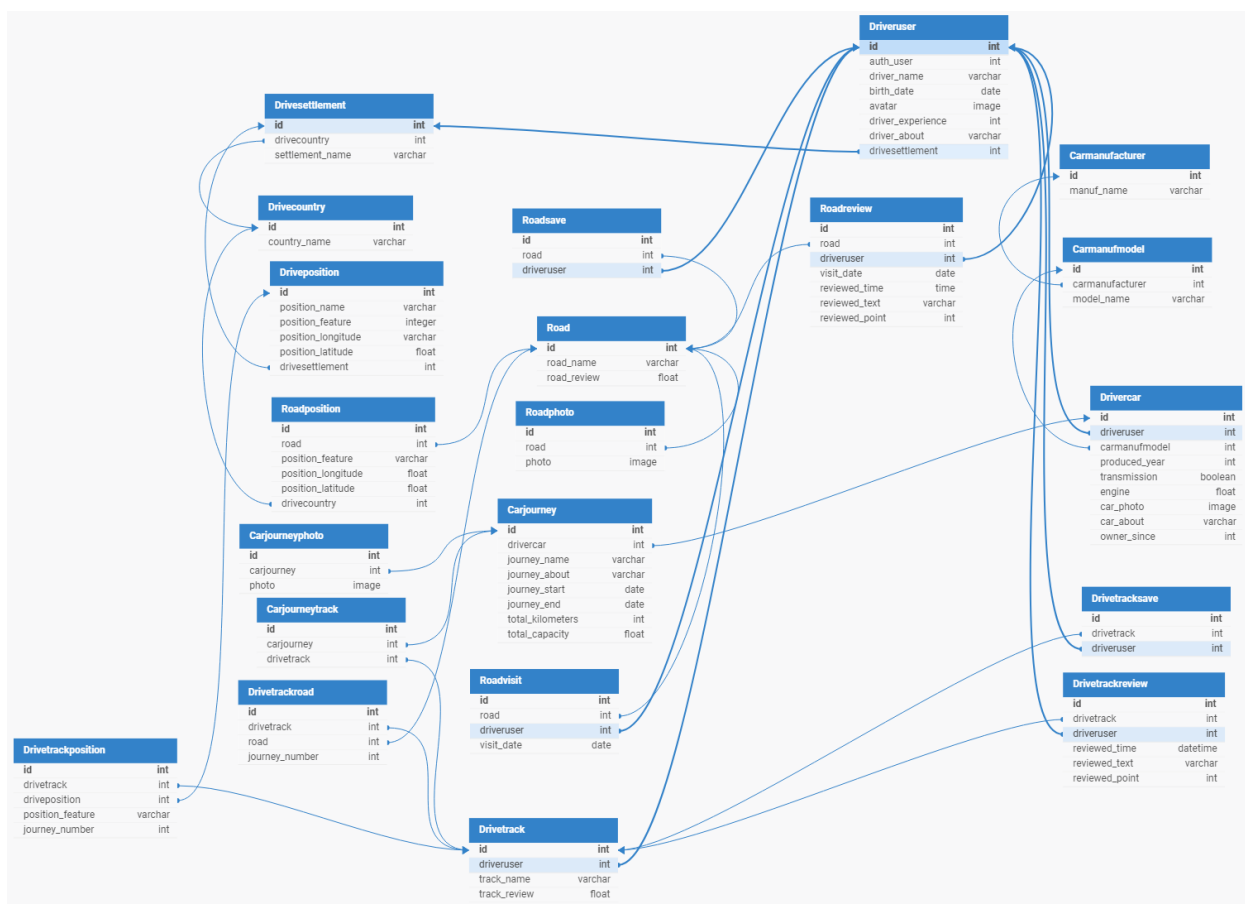


Рисунок 2.2 – Схема БД програми створення відгуків про самостійні подорожі автомобілем

Збереження широковідомих доріг користувачем задаються таблицею Roadsaver, визначеною полями:

- id – номер, що унікально ідентифікує збереження широковідомої дороги користувачем;

- road – широковідома дорога, яка зберігається користувачем (це зв'язуюче поле, що належить до таблиці Road);
- driveruser – зареєстрований користувач, який зберіг цю широковідому дорогу (це зв'язуюче поле, що належить до таблиці Driveruser).

2.4 Висновки за розділом 2

Основні результати розділу охоплюють вибір технологій розробки, куди входять мова Python з вебфреймворком Django, визначення сценаріїв роботи з програмою користувача як основних, так і альтернативних, визначення структури БД.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Визначення порядку роботи з програмою

Порядок роботи з програмою продиктований користувацькою взаємодією, коли користувач визначає дію з меню, а вона виконується програмою (рис. 3.1).

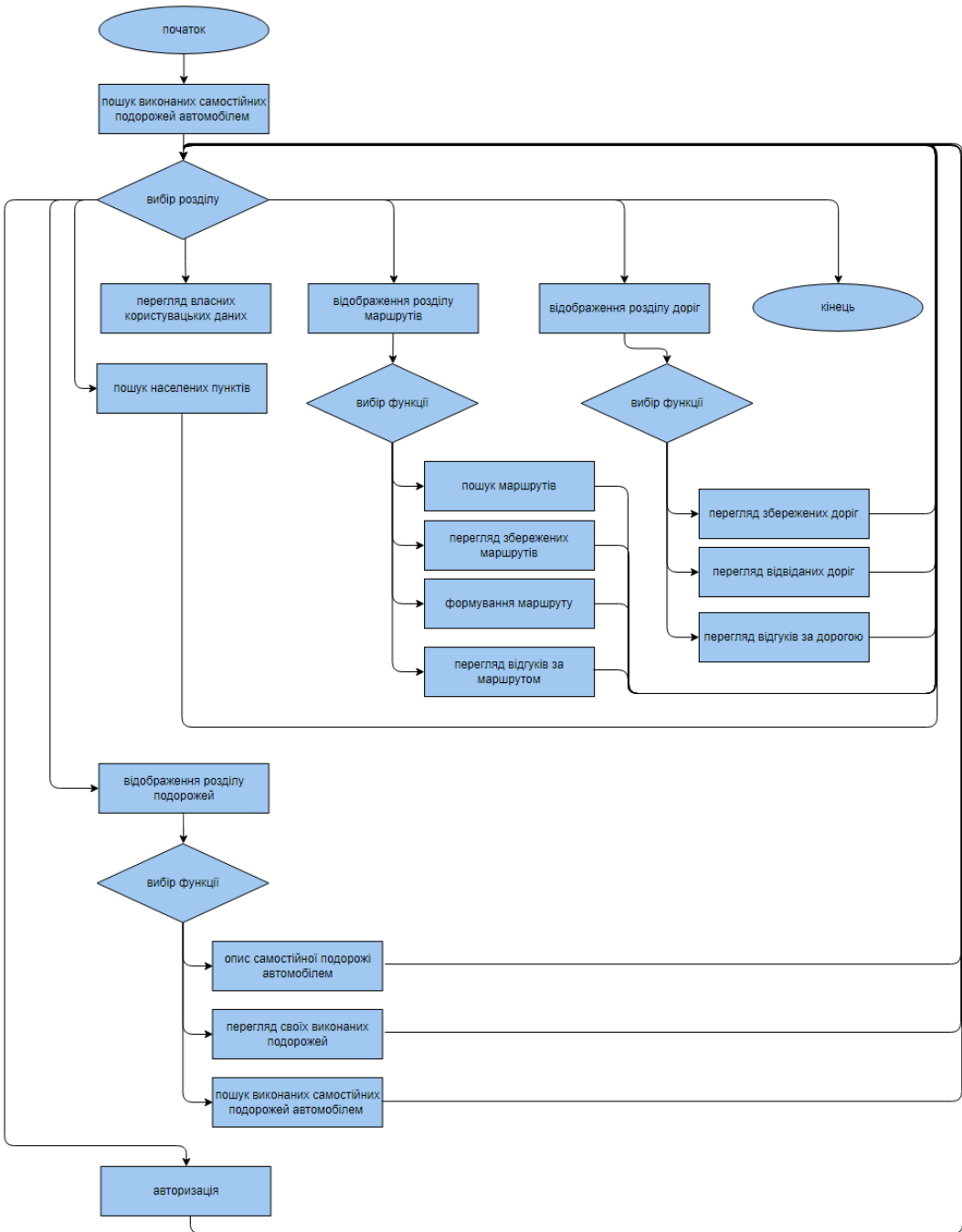


Рисунок 3.1 – Схема функціонування програми створення відгуків про самостійні подорожі автомобілем

3.2 Результати реалізації програмних одиниць додатку

Програмні одиниці додатку створені з забезпеченням шаблону Model-View-Template (MVT), який характерний для Django, але оскільки при цьому представлення використовують шаблони, то вони описані нижче разом, а вже далі описано моделі.

Перегляд запису широковідомої дороги реалізовано функцією `road_view`, для доступу до якої необхідно задати параметр `road_num`, який встановлює номер цієї дороги. Ця функція реалізує представлення в шаблоні MVT. За номером визначається об'єкт самої дороги за класом `Road`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що “Така дорога відсутня, введено некоректний ідентифікатор”. Якщо дорогу знайдено, то за цим об'єктом виділяються фотографії, пов'язані з дорогою, місця на мапі, які входять до цієї дороги, 10 найбільш рейтингових (з найвищими середніми оцінками за відгуками) маршрутів, що включають цю дорогу, 10 останніх самостійних подорожей автомобілем користувачів, які включають цю дорогу, 10 останніх відгуків про цю дорогу та інформація щодо дороги, пов'язана з даними авторизованого користувача, якщо користувач авторизувався: визначення того, чи створював відгук за цією дорогою цей користувач, чи зберігав її як відвідану, чи зберіг ці маршрути для себе користувач. Результати вносяться в шаблон сторінки `road.html`.

Перегляд всіх відгуків за широковідомою дорогою реалізовано функцією `all_road_reviews_view`, для доступу до якої необхідно задати параметр `road_num`, який встановлює номер цієї дороги. За номером визначається об'єкт самої дороги за класом `Road`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що “Така дорога відсутня, введено некоректний ідентифікатор”. Якщо дорогу знайдено, то виділяються 3 фотографії цієї дороги, всі відгуки за нею з впорядкуванням з останнього та інформація,

пов'язана з даними авторизованого користувача, якщо користувач авторизувався: чи зберіг цю дорогу собі цей користувач. Результати вносяться в шаблон сторінки `roadreviews.html`.

Перегляд всіх подорожей за широковідомою дорогою реалізовано функцією `all_road_journeys_view`, для доступу до якої необхідно задати параметр `road_num`, який встановлює номер цієї дороги. За номером визначається об'єкт самої дороги за класом `Road`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що "Така дорога відсутня, введено некоректний ідентифікатор". Якщо дорогу знайдено, то виділяються 3 фотографії цієї дороги, всі подорожі, маршрути яких включають цю дорогу, впорядковуючи їх від останньої, та інформація, пов'язана з даними авторизованого користувача, якщо користувач авторизувався: чи зберіг цю дорогу собі цей користувач. Результати вносяться в шаблон сторінки `roadjourneys.html`.

Перегляд маршруту самостійної подорожі автомобілем реалізовано функцією `drivetrack_view`, для доступу до якої необхідно задати параметр `drivetrack_num`, який встановлює номер цього маршруту. За номером визначається об'єкт самого маршруту за класом `Drivetrack`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що "Такий маршрут самостійної подорожі відсутній, введено некоректний ідентифікатор". Якщо маршрут знайдено, то повертається сам об'єкт маршруту самостійної подорожі автомобілем, позиції цього маршруту, які включають місця на мапі, широковідомі дороги, які входять до маршруту, 10 останніх відгуків про маршрут, 10 останніх самостійних подорожей, які включають цей маршрут, а також інформація, пов'язана з даними авторизованого користувача, якщо користувач авторизувався: чи зберіг цей маршрут самостійної подорожі собі цей користувач. Результати вносяться в шаблон сторінки `drivetrack.html`.

Перегляд всіх відгуків за маршрутом самостійної подорожі автомобілем реалізовано функцією `all_drivetrack_reviews_view`, для доступу до

якої необхідно задати параметр `drivetrack_num`, який встановлює номер цього маршруту. За номером визначається об'єкт самого маршруту за класом `Drivetrack`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що “Такий маршрут самостійної подорожі відсутній, введено некоректний ідентифікатор”. Якщо маршрут знайдено, то повертається об'єкт самого маршруту, всі відгуки про цей маршрут, а також інформація, пов'язана з даними авторизованого користувача, якщо користувач авторизувався: чи зберіг цей маршрут самостійної подорожі собі цей користувач. Результати вносяться в шаблон сторінки `drivetrackreviews.html`.

Перегляд всіх подорожей за маршрутом самостійної подорожі автомобілем реалізовано функцією `all_drivetrack_journeys_view`, для доступу до якої необхідно задати параметр `drivetrack_num`, який встановлює номер цього маршруту. За номером визначається об'єкт самого маршруту за класом `Drivetrack`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що “Такий маршрут самостійної подорожі відсутній, введено некоректний ідентифікатор”. Якщо маршрут знайдено, то повертається об'єкт самого маршруту, всі самостійні подорожі автомобілем користувачів, які включали цей маршрут до свого складу, а також інформація, пов'язана з даними авторизованого користувача, якщо користувач авторизувався: чи зберіг цей маршрут самостійної подорожі собі цей користувач. Результати вносяться в шаблон сторінки `drivetrackjourneys.html`.

Перегляд збережених доріг реалізовано функцією `all_saved_roads_view`. Якщо користувач не авторизований, то повертається сторінка з повідомленням про помилку “Збереження доступні тільки після авторизації”. Якщо користувач авторизувався, то вибираються всі збережені ним дороги, за кожною дорогою – одна фотографія, кількість відгуків, отриманих щодо цієї дороги, позначка про те, чи позначав цю дорогу як

відвідану користувач. Результати вносяться в шаблон сторінки savedroads.html.

Перегляд збережених маршрутів самостійних подорожей реалізовано функцією all_saved_drivetracks_view. Якщо користувач не авторизований, то повертається сторінка з повідомленням про помилку “Збереження доступні тільки після авторизації”. Якщо користувач авторизувався, то вибираються всі збережені ним маршрути самостійних подорожей, за кожним маршрутом – кількість відгуків, отриманих щодо цього маршруту. Результати вносяться в шаблон сторінки saveddrivetracks.html. Шаблони реалізовані з макетуванням сторінки початково на основі шаблону [8].

Перегляд всіх самостійних подорожей автомобілем користувача реалізовано функцією driveruser_journeys_view, для доступу до якої необхідно задати параметр driver_user_name, що вказує логін користувача, але це значення може бути і пропущено. Якщо значення пропущено, то сторінка формується за даними, що відповідають авторизації. Відбираються всі самостійні подорожі автомобілем, які описав користувач. За кожною подорожжю відбираються фотографії. Результати вносяться в шаблон сторінки driverjourneys.html. Якщо користувач не авторизувався, а значення параметра не задано, то повертається сторінка з повідомленням “Потрібно або авторизуватися, або задати користувача”.

Перегляд всіх відвіданих користувачем доріг реалізовано функцією driveruser_visited_roads_view, для доступу до якої необхідно задати параметр driver_user_name, що вказує логін користувача, але це значення може бути і пропущено. Якщо значення пропущено, то сторінка формується за даними, що відповідають авторизації. Відбираються всі дороги, які відвідав користувач. Дані про відвідані дороги збираються з 3 джерел:

- відгуків про широковідомі дороги цього користувача;
- визначень відвідування доріг цим користувачем;
- маршрутів описаних самостійних подорожей автомобілем цього користувача, які включають цей маршрут до свого складу.

За кожною дорогою відбирається одна фотографія, визначається, чи відвідував її користувач та яку кількість відгуків вона має. Результати вносяться в шаблон сторінки `visitedroads.html`. Якщо користувач не авторизувався, а значення параметра не задано, то повертається сторінка з повідомленням “Потрібно або авторизуватися, або задати користувача”.

Перегляд відгуків користувача про маршрути самостійної подорожі реалізовано функцією `driveuser_drivetrack_reviews_view`, для доступу до якої необхідно задати параметр `driver_user_name`, що вказує логін користувача, але це значення може бути і пропущено. Якщо значення пропущено, то сторінка формується за даними, що відповідають авторизації. Відбираються всі відгуки, які створені цим користувачем про маршрути самостійної подорожі, впорядковуючи за датою відгуку з останніх. За кожним маршрутом, якого стосується відгук, перевіряється, чи зберіг його користувач собі. Результати вносяться в шаблон сторінки `dudrivetrackreviews.html`. Якщо користувач не авторизувався, а значення параметра не задано, то повертається сторінка з повідомленням “Потрібно або авторизуватися, або задати користувача”.

Перегляд відгуків користувача про дороги реалізовано функцією `driveuser_road_reviews_view`, для доступу до якої необхідно задати параметр `driver_user_name`, що вказує логін користувача, але це значення може бути і пропущено. Якщо значення пропущено, то сторінка формується за даними, що відповідають авторизації. Відбираються всі відгуки, які створені цим користувачем про дороги, впорядковуючи за датою візиту з останніх. За кожною дорогою, якої стосується відгук, перевіряється, чи зберіг її користувач собі. Результати вносяться в шаблон сторінки `duroadreviews.html`. Якщо користувач не авторизувався, а значення параметра не задано, то повертається сторінка з повідомленням “Потрібно або авторизуватися, або задати користувача”.

Перегляд запису населеного пункту реалізовано функцією `drivesettlement_view`, для доступу до якої необхідно задати параметр `drivesettlement_num`, що задає номер населеного пункту. За номером

визначається об'єкт населеного пункту за класом `Drivesettlement`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що “Такий населений пункт відсутній, введено некоректний ідентифікатор”. Якщо населений пункт знайдено, то повертається об'єкт цього населеного пункту, маршрути самостійних подорожей, до яких входить цей населений пункт. Результати вносяться в шаблон сторінки `settlement.html`.

Перегляд виконаної самостійної подорожі автомобілем реалізовано функцією `carjourney_view`, для доступу до якої необхідно задати параметр `carjourney_num`, що задає номер виконаної самостійної подорожі автомобілем. За номером визначається об'єкт виконаної самостійної подорожі автомобілем за класом `Carjourney`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що “Така самостійна подорож автомобілем відсутня, введено некоректний ідентифікатор”. Якщо самостійну подорож автомобілем знайдено, то повертається об'єкт цієї самостійної подорожі, фотографії, віднесені автором до неї, усі маршрути, які були охоплені цією подорожжю, а за кожним маршрутом – його склад відносно тих місць на мапі, які входять у маршрут, доріг, які охоплює цей маршрут. Результати вносяться в шаблон сторінки `carjourney.html`.

Створення відгуку про маршрут самостійної подорожі реалізовано функцією `drivetrackreview_publish_view`, для доступу до якої необхідно задати параметр `drivetrack_num`, що задає номер маршруту самостійної подорожі. За номером визначається об'єкт маршрут самостійної подорожі за класом `Drivetrack`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що “Такий маршрут самостійної подорожі автомобілем відсутній, введено некоректний ідентифікатор”. Якщо об'єкт знайдений, то повертається форма класу `DrivetrackreviewForm`. Дані об'єкту маршруту самостійної подорожі вносяться

в кодї, також виконуються і дії, пов'язані з внесенням автора відгуку як авторизованого користувача.

Створення відгуку про про широківідому дорогу реалізовано функцією `roadreview_publish_view`, для доступу до якої необхідно задати параметр `road_num`, що задає номер широківідомої дороги. За номером визначається об'єкт широківідомої дороги за класом `Road`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що "Така дорога відсутня, введено некоректний ідентифікатор". Якщо об'єкт знайдений, то повертається форма класу `RoadreviewForm`. Дані об'єкту широківідомої дороги вносяться в кодї, також виконуються і дії, пов'язані з внесенням автора відгуку як авторизованого користувача.

Формування маршруту, який може використовуватися для самостійної подорожі автомобілем, реалізовано функцією `drivetrack_create_view`. Підтримується ця функція через форму `DrivepositionForm`, що дозволяє за необхідності створити потрібне місце на мапі, якщо воно не було задано раніше, при цьому на основі цієї форми може накопичуватися ряд позицій або обиратися з існуючих, для чого в представлення передається набір всіх об'єктів моделі `Driveposition` та моделі `Road`. У підсумку при фінальному створенні маршруту додаються зв'язки з місцями на мапі та дорогами.

Перегляд даних користувача реалізовано функцією `driveruser_view`, для доступу до якої необхідно задати параметр `driver_user_name`, що вказує логін користувача, але це значення може бути і пропущено. Якщо значення пропущено, то сторінка формується за даними, що відповідають авторизації. Після визначення користувача витягаються дані об'єкта користувача з моделі `Driveruser`, пов'язані з користувачем дані автомобілів з моделі `Drivercar`, дані самостійних подорожей, опис яких здійснив користувач, з моделі `Carjourney`. Результати вносяться в шаблон сторінки `driveruser.html`. Редагування даних користувача реалізовано функцією `driveruser_edit_view`.

Додавання інформації про використовуваний автомобіль користувачем для самостійних подорожей реалізовано функцією `drivercar_create_view`,

робота якої базується на формі `DrivercarForm`, що дозволяє створити відповідний об'єкт та в коді пов'язати його з зареєстрованим користувачем як власником автомобіля. Редагування інформації про використовуваний автомобіль користувачем для самостійних подорожей забезпечується функцією `drivercar_edit_view`, де задається вже існуючий автомобіль, а при спробі зберегти внесені результати, а також передати їх для редагування відбувається перевірка, що цей автомобіль належить користувачу, який авторизувався.

Збереження широковідомої дороги реалізовано функцією `road_save_view`, для доступу до якої необхідно задати параметр `road_num`, який встановлює номер цієї дороги. За номером визначається об'єкт самої дороги за класом `Road`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що “Така дорога відсутня, введено некоректний ідентифікатор”. Якщо дорогу знайдено, то для користувача, який авторизувався, вноситься дана дорога в перелік збережених. Але попередньо перевіряється, чи не присутня ця дорога у переліку збережених цим користувачем вже. Якщо це виконується, то реалізується вилучення збереженої широковідомої дороги.

Збереження маршруту самостійної подорожі реалізовано функцією `drivetrack_save_view`, для доступу до якої необхідно задати параметр `drivetrack_num`, який встановлює номер цього маршруту самостійної подорожі. За номером визначається об'єкт самого маршруту самостійної подорожі за класом `Drivetrack`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що “Такий маршрут самостійної подорожі відсутній, введено некоректний ідентифікатор”. Якщо маршрут самостійної подорожі знайдено, то для користувача, який авторизувався, вноситься даний маршрут самостійної подорожі в перелік збережених. Але попередньо перевіряється, чи не присутній цей маршрут у переліку збережених цим користувачем вже. Якщо

це виконується, то реалізується вилучення збереженого маршруту самостійної подорожі.

Визначення відвідування дороги реалізовано функцією `road_visit_view`, для доступу до якої необхідно задати параметр `road_num`, який встановлює номер цієї дороги. За номером визначається об'єкт самої дороги за класом `Road`, а якщо такий об'єкт не існує, то повертається вебсторінка з помилкою, де зазначається повідомлення, в якому вказується, що “Така дорога відсутня, введено некоректний ідентифікатор”. Якщо дорогу знайдено, то для користувача, який авторизувався, вносяться дані про відвідування дороги ним як об'єкт класу моделі `Roadvisit`.

Пошук маршрутів самостійних подорожей автомобілем реалізовано функцією `drivetrack_search_view`. Пошук виконується за допомогою параметрів, які користувач вводить в окрему форму, а значення параметрів розбираються як параметри GET-запиту, включаючи:

- назву маршруту самостійної подорожі автомобілем;
- дорогу, через яку має проходити маршрут самостійної подорожі автомобілем;
- населений пункт, через який має проходити маршрут самостійної подорожі автомобілем;
- країну, через яку має проходити маршрут самостійної подорожі автомобілем;
- мінімальну середню оцінку, яку повинен мати маршрут самостійної подорожі автомобілем.

Результати відфільтрування маршрутів самостійних подорожей за заданими параметрами з моделі `Drivetrack` вносяться в шаблон сторінки `drivetracksearch.html`, де і задаються самі параметри.

Пошук населених пунктів реалізовано функцією `drivesettlement_view`. Пошук виконується за допомогою параметрів, які користувач вводить в окрему форму, а значення параметрів розбираються як параметри GET-запиту, включаючи:

- назву населеного пункту;
- країну, де має знаходитися населений пункт.

Результати відфільтрування населених пунктів за заданими параметрами з моделі `Drivesettlement` вносяться в шаблон сторінки `settlementsearch.html`, де і задаються самі параметри.

Пошук виконаних самостійних подорожей автомобілем реалізовано функцією `carjourney_search_view`. Пошук виконується за допомогою параметрів, які користувач вводить в окрему форму, а значення параметрів розбираються як параметри GET-запиту, включаючи:

- назву виконаної самостійної подорожі автомобілем;
- початок періоду, в який мала відбутися самостійна подорож автомобілем;
- закінчення періоду, в який мала відбутися самостійна подорож автомобілем;
- мінімальну тривалість подорожі в кілометрах;
- максимальну тривалість подорожі в кілометрах;
- дорогу, через яку має проходити маршрут самостійної подорожі автомобілем, який використовувався під час цієї подорожі;
- населений пункт, через який має проходити маршрут самостійної подорожі автомобілем, який використовувався під час цієї подорожі;
- країну, через яку має проходити маршрут самостійної подорожі автомобілем, який використовувався під час цієї подорожі.

Опис виконаної самостійної подорожі автомобілем реалізовано функцією `carjourney_create_view`, яка базується на класі форми `CarjourneyForm`. При цьому додатково відбувається збереження фотографій, пов'язаних під час роботи з формою з цією виконаною самостійною подорожжю автомобілем, маршрутів, які використовувалися під час подорожі. Тобто в даному представленні самі маршрути не створюються, вони заздалегідь мають бути створені на основі описаного вище представлення. Редагування опису виконаної самостійної подорожі автомобілем відповідно

також підтримується тією самою формою, але виконується перевірка того, що ця подорож здійснювалась автомобілем, який використовується користувачем, який авторизувався в програмі. Функція називається `carjourney_edit_view`.

Далі представлені моделі шаблону MVT з даного вебдодатку.

Клас `Driveposition` програмно визначає модель місця на мапі наслідуванням `django.db.models.Model`.

Поле `position_name` реалізовано типом `CharField` для визначення назви місця на мапі.

Поле `position_feature` реалізовано типом `CharField` для визначення коментаря до особливості цього місця.

Поле `position_longitude` реалізовано типом `FloatField` для визначення довготи точки на мапі цього місця.

Поле `position_latitude` реалізовано типом `FloatField` для визначення широти точки на мапі цього місця.

Поле `drivesettlement` реалізовано типом `ForeignKey` для визначення населеного пункту, в межах якого знаходиться це місце на мапі, через модель `Drivesettlement`.

Клас `Drivesettlement` програмно визначає модель населеного пункту наслідуванням `django.db.models.Model`.

Поле `drivecountry` реалізовано типом `ForeignKey` для визначення країни, в якій знаходиться населений пункт, через модель `Drivecountry`.

Поле `settlement_name` реалізовано типом `CharField` для визначення назви населеного пункту.

Клас `Drivecountry` програмно визначає модель країни, з якою працює програма і яка може використовуватися в маршрутах самостійних подорожей автомобілем, наслідуванням `django.db.models.Model`, задаючи додатково тільки поле `country_name`, що реалізовано типом `CharField` для визначення назви країни, що може використовуватися в маршрутах самостійних подорожей автомобілем.

Клас `Drivetrack` програмно визначає модель маршруту, який може використовуватися для самостійної подорожі автомобілем, наслідуванням `django.db.models.Model`.

Поле `driveruser` реалізовано типом `ForeignKey` для визначення водія, який є зареєстрованим користувачем та який створив цей маршрут у програмі, через модель `Driveruser`.

Поле `track_name` реалізовано типом `CharField` для визначення назви маршруту, який може використовуватися для самостійної подорожі автомобілем.

Поле `track_review` реалізовано типом `FloatField` для визначення оцінки, яку на даний момент усереднено отримав цей маршрут, який може використовуватися для самостійної подорожі автомобілем.

Клас `Drivetrackposition` програмно визначає модель конкретного місця на мапі, яке входить до складу маршруту самостійної подорожі автомобілем, наслідуванням `django.db.models.Model`.

Поле `drivetrack` реалізовано типом `ForeignKey` для визначення маршруту, який може використовуватися для самостійної подорожі автомобілем і до складу якого входить це місце на мапі, через модель `Drivetrack`.

Поле `driveposition` реалізовано типом `ForeignKey` для визначення місця на мапі, яке входить до маршруту, який може використовуватися для самостійної подорожі автомобілем, через модель `Driveposition`.

Поле `position_feature` реалізовано типом `CharField` для визначення особливості цього місця на мапі у складі даного маршруту, який може використовуватися для самостійної подорожі автомобілем.

Поле `journey_number` реалізовано типом `PositiveSmallIntegerField` для визначення позиції цього місця на мапі в порядку маршруту, який може використовуватися для самостійної подорожі автомобілем: це позиція не тільки серед всіх місць на мапі у складі маршруту, але і серед доріг, які також можуть бути додані до маршруту.

Клас `Drivetrackroad` програмно визначає модель входження дороги до складу маршруту наслідуванням `django.db.models.Model`.

Поле `drivetrack` реалізовано типом `ForeignKey` для визначення маршруту, який може використовуватися для самостійної подорожі автомобілем і до складу якого входить ця широковідома , через модель `Drivetrack`.

Поле `road` реалізовано типом `ForeignKey` для визначення широковідомої дороги, яка входить до маршруту, який може використовуватися для самостійної подорожі, через модель `Road`.

Поле `journey_number` реалізовано типом `PositiveSmallIntegerField` для визначення позиції цієї дороги в порядку маршруту, який може використовуватися для самостійної подорожі автомобілем: це позиція не тільки серед всіх доріг, але і серед місць на мапі у складі маршруту, які також можуть бути додані до маршруту.

Клас `Road` програмно визначає модель широковідомих доріг, по яким здійснюють самостійні подорожі автомобілем, наслідуванням `django.db.models.Model`.

Поле `road_name` реалізовано типом `CharField` для визначення назви широковідомої дороги, якою здійснюють самостійні подорожі автомобілем;

Поле `road_review` реалізовано типом `FloatField` для визначення оцінки, яку на даний момент усереднено отримала ця широковідома дорога.

Клас `Roadphoto` програмно визначає модель фотографій широковідомих доріг наслідуванням `django.db.models.Model`.

Поле `road` реалізовано типом `ForeignKey` для визначення широковідомої дороги, подорож якою зображена на фотографії, через модель `Road`.

Поле `photo` реалізовано типом `ImageField` для визначення самої фотографії широковідомої дороги.

Клас `Roadposition` програмно визначає модель місця на мапі, що визначає позицію широковідомої дороги, наслідуванням `django.db.models.Model`.

Поле `road` реалізовано типом `ForeignKey` для визначення широковідомої дороги, яка включає це місце на мапі, через модель `Road`.

Поле `position_feature` реалізовано типом `CharField` для визначення особливості цього місця на мапі стосовно даної широковідомої дороги.

Поле `position_longitude` реалізовано типом `FloatField` для визначення довготи точки на мапі цього місця з широковідомої дороги.

Поле `position_latitude` реалізовано типом `FloatField` для визначення широти точки на мапі цього місця з широковідомої дороги.

Поле `drivecountry` реалізовано типом `ForeignKey` для визначення країни, де це місце знаходиться, через модель `Drivecountry`.

Клас `Driveruser` програмно визначає модель зареєстрованого користувача наслідуванням `django.db.models.Model`.

Поле `auth_user` реалізовано типом `ForeignKey` для визначення облікового запису з Django, який використовується зареєстрованим користувачем, через модель `User`.

Поле `driver_name` реалізовано типом `CharField` для визначення імені водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем.

Поле `birth_date` реалізовано типом `DateField` для визначення дати народження водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем.

Поле `avatar` реалізовано типом `ImageField` для визначення фотографії водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем.

Поле `driver_experience` реалізовано типом `PositiveSmallIntegerField` для визначення досвіду водіння водія, який здійснює самостійні подорожі

автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем.

Поле `driver_about` реалізовано типом `CharField` для визначення інформації про водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем.

Поле `drivesettlement` реалізовано типом `ForeignKey` для визначення населеного пункту, в якому проживає водій, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем, через модель `Drivesettlement`.

Поле `drivetracks` реалізовано типом `ManyToManyField` для визначення збережених маршрутів цим зареєстрованим користувачем.

Поле `roads` реалізовано типом `ManyToManyField` для визначення збережених широковідомих доріг цим зареєстрованим користувачем.

Клас `Drivercar` програмно визначає модель автомобіля водія, який здійснює самостійні подорожі автомобілем або цікавиться такими подорожами, є зареєстрованим користувачем, наслідуванням `django.db.models.Model`.

Поле `driveruser` реалізовано типом `ForeignKey` для визначення зареєстрованого користувача, який є водієм цього автомобіля, через модель `Driveruser`.

Поле `carmanufmodel` реалізовано типом `ForeignKey` для визначення моделі автомобіля, яким користується водій, який є зареєстрованим користувачем, для самостійних подорожей автомобілем, через модель `Carmanufacturer`.

Поле `produced_year` реалізовано типом `PositiveSmallIntegerField` для визначення року виробництва автомобіля, яким користується водій, який є зареєстрованим користувачем, для самостійних подорожей автомобілем.

Поле `transmission` реалізовано типом `BooleanField` для визначення тип коробки передач, який використовується в автомобілі (автоматична,

механічна), що користується водієм, який є зареєстрованим користувачем, для самостійних подорожей автомобілем.

Поле `engine` реалізовано типом `FloatField` для визначення об'єму двигуна автомобіля, яким користується водій, який є зареєстрованим користувачем, для самостійних подорожей автомобілем.

Поле `car_photo` реалізовано типом `ImageField` для визначення фотографії автомобіля, яким користується водій, який є зареєстрованим користувачем, для самостійних подорожей автомобілем.

Поле `car_about` реалізовано типом `CharField` для визначення опису автомобіля, яким користується водій, який є зареєстрованим користувачем, для самостійних подорожей автомобілем.

Поле `owner_since` реалізовано типом `PositiveSmallIntegerField` для визначення року, з якого є власником цього автомобіля (або фактично використовує його) цей зареєстрований користувач.

Клас `Drivercar` програмно визначає виробників автомобілів, які доступні в програмі, наслідуванням `django.db.models.Model`. Додається тільки одне поле `manuf_name`, реалізоване типом `CharField` для визначення назви виробника автомобіля, доступного в програмі.

Клас `Drivercar` програмно визначає моделі виробників автомобілів, які доступні в програмі, наслідуванням `django.db.models.Model`.

Поле `carmanufacturer` реалізовано типом `ForeignKey` для визначення виробника, який виготовляє цю модель автомобіля, доступну в програмі, через модель `Carmanufacturer`.

Поле `model_name` реалізовано типом `CharField` для визначення назви моделі автомобіля, яка доступна в програмі.

Клас `Carjourney` програмно визначає моделі самостійних подорожей автомобілем, які вже здійснив зареєстрований користувач, наслідуванням `django.db.models.Model`.

Поле `drivercar` реалізовано типом `ForeignKey` для визначення автомобіля водія, який є зареєстрованим користувачем, яким було здійснено цю самостійну подорож автомобілем, через модель `Drivercar`.

Поле `journey_name` реалізовано типом `CharField` для визначення назви самостійної подорожі автомобілем, яку було здійснено.

Поле `journey_about` реалізовано типом `CharField` для визначення опису самостійної подорожі автомобілем, яку було здійснено.

Поле `journey_start` реалізовано типом `DateField` для визначення дати початку самостійної подорожі автомобілем, яку було здійснено.

Поле `journey_end` реалізовано типом `DateField` для визначення дати закінчення самостійної подорожі автомобілем, яку було здійснено.

Поле `total_kilometers` реалізовано типом `PositiveIntegerField` для визначення загальної довжини самостійної подорожі автомобілем, яку було здійснено, в кілометрах.

Поле `total_capacity` реалізовано типом `FloatField` для визначення загальної витрати палива на самостійну подорож автомобілем, яку було здійснено.

Поле `drivetracks` реалізовано типом `ManyToManyField` для визначення маршрутів, які охоплювала ця самостійна подорож автомобілем, що дозволило реалізувати таблицю `Carjourneytrack`.

Клас `Carjourneyphoto` програмно визначає моделі фотографій самостійної подорожі автомобілем, яку було здійснено, наслідуванням `django.db.models.Model`.

Поле `carjourney` реалізовано типом `ForeignKey` для визначення самостійної подорожі автомобілем, до якої належить ця фотографія, через модель `Carjourney`.

Поле `photo` реалізовано типом `ImageField` для визначення самої фотографія з самостійної подорожі автомобілем.

Клас `Drivetrackreview` програмно визначає моделі відгуків на маршрути самостійних подорожей автомобілем наслідуванням `django.db.models.Model`.

Поле `drivetrack` реалізовано типом `ForeignKey` для визначення маршруту самостійної подорожі автомобілем, на який створений цей відгук, через модель `Drivetrack`.

Поле `driveruser` реалізовано типом `ForeignKey` для визначення зареєстрованого користувача, що створив цей відгук на маршрут самостійної подорожі автомобілем, через модель `Driveruser`.

Поле `reviewed_time` реалізовано типом `DateTimeField` для визначення моменту створення відгуку на маршрут самостійної подорожі автомобілем.

Поле `reviewed_text` реалізовано типом `CharField` для визначення тексту відгуку на маршрут самостійної подорожі автомобілем.

Поле `reviewed_point` реалізовано типом `PositiveSmallIntegerField` для визначення оцінки з відгуку на маршрут самостійної подорожі автомобілем.

Клас `Roadreview` програмно визначає моделі відгуків на самостійні подорожі широковідомими дорогами наслідуванням `django.db.models.Model`.

Поле `road` реалізовано типом `ForeignKey` для визначення широковідомої дороги, на самостійну подорож якою створив відгук користувач, через модель `Road`.

Поле `driveruser` реалізовано типом `ForeignKey` для визначення зареєстрованого користувача, який подорожував самостійно автомобілем цією широковідомою дорогою і створив цей відгук, через модель `Driveruser`.

Поле `visit_date` реалізовано типом `DateField` для визначення дати, коли цей зареєстрований користувач подорожував самостійно автомобілем цією широковідомою дорогою.

Поле `reviewed_time` реалізовано типом `DateTimeField` для визначення дати, коли було створено цей відгук на самостійну подорож широковідомою дорогою.

Поле `reviewed_text` реалізовано типом `CharField` для визначення тексту відгуку на самостійну подорож широковідомою дорогою.

Поле `reviewed_point` реалізовано типом `PositiveSmallIntegerField` для визначення оцінки відгуку на самостійну подорож широковідомою дорогою.

Клас Roadvisit програмно визначає моделі відвідування широковідомої дороги наслідуванням `django.db.models.Model`.

Поле `road` реалізовано типом `ForeignKey` для визначення широковідомої дороги, яку було відвідано користувачем, через модель `Road`.

Поле `driveruser` реалізовано типом `ForeignKey` для визначення користувача, який відвідав цю широковідому дорогу, через модель `Driveruser`.

Поле `visit_date` реалізовано типом `DateField` для визначення дати відвідування широковідомої дороги користувачем.

3.3 Висновки за розділом 3

Реалізацію вебдодатку виконано за шаблоном MVT, що відповідає Django, з визначенням всіх реалізованих програмних одиниць за тими функціональними вимогами, які були встановлені під час проєктування і моделювання, включаючи сукупності шаблонів вебсторінок і представлень, які наповнюють ці шаблони даними, моделі.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Призначення програми в тому, щоб підтримувати самостійні подорожі автомобілем, дозволяючи надавати інформацію про досвід інших користувачів, шукати нові варіанти для подорожей, ділитися власним досвідом.

4.2 Умови виконання програми

Для експлуатації програми створення відгуків про самостійні подорожі автомобілем обов'язково потрібно виділити адміністратора програми або користувача, здатного виконувати такі функції. В обов'язки адміністратора входить наповнення даними програми зокрема стосовно населених пунктів, широковідомих доріг. Також адміністратор повинен виконати розгортання додатку на сервері. Технічне забезпечення сервера повинно включати мінімально процесор Intel Xeon E5440 (частота 2,83 ГГц), жорсткий диск – 100 Гб (потрібно враховувати вимоги до збереження фотографій, яких потенційно може бути достатньо багато і розширювати простір), оперативна пам'ять – 4 Гб.

4.3 Виконання і тестування програми

Над програмою було проведено тестування, яке з одного боку мало перевірити загальну придатність програми до роботи, відсутність випадків випадання, випадкового завершення роботи, а з іншого – перевірити забезпечення очікуваного виконання функцій за сценаріями з розділу 2. Перевірка виявилась повністю успішною. Для деталізації процесу роботи з програмою далі описано логіку роботи та приклади отриманих результатів взаємодії.

Стандартним початком роботи є сторінка пошуку виконаних самостійних подорожей автомобілем. Під час пошуку можна задати один або декілька з наступних параметрів:

- назву виконаної самостійної подорожі автомобілем;
- початок періоду, в який мала відбутися самостійна подорож автомобілем;
- закінчення періоду, в який мала відбутися самостійна подорож автомобілем;
- мінімальну тривалість подорожі в кілометрах;
- максимальну тривалість подорожі в кілометрах;
- дорогу, через яку має проходити маршрут самостійної подорожі автомобілем, який використовувався під час цієї подорожі;
- населений пункт, через який має проходити маршрут самостійної подорожі автомобілем, який використовувався під час цієї подорожі;
- країну, через яку має проходити маршрут самостійної подорожі автомобілем, який використовувався під час цієї подорожі.

Після цього можна відкрити той опис виконаної самостійної подорожі автомобілем, яка потрібна (рис. 4.1). На тій сторінці буде продемонстровано основні дані щодо подорожі, які включають назву, автора, модель та рік випуску його автомобіля (можна натиснути на фотографію користувача та відкрити його сторінку), набір фотографій з подорожі (якщо фотографій декілька, то показуються стрілки для перемикання між ними), дату початку подорожі, дату закінчення, кілометраж подорожі та обсяг витраченого палива на неї. Далі розташовуються опис текстом виконаної самостійно подорожі, маршрут або маршрути, які охоплювала подорож.

З розділу меню “Подорожі” можна також обрати пошук, але окрім того ще перейти до створення опису нової виконаної самостійно подорожі автомобілем, перегляду власних виконаних подорожей автомобілем.



Рисунок 4.1 – Сторінка опису виконаної самостійної подорожі автомобілем

З розділу меню “Маршрути” можна перейти до пошуку потрібного маршруту, до перегляду збережених маршрутів або до додавання нового маршруту, якого ще немає в програмі.

Пошук маршрутів самостійних подорожей автомобілем включає визначення таких параметрів:

- назву маршруту самостійної подорожі автомобілем;
- дорогу, через яку має проходити маршрут самостійної подорожі автомобілем;
- населений пункт, через який має проходити маршрут самостійної подорожі автомобілем;
- країну, через яку має проходити маршрут самостійної подорожі автомобілем;
- мінімальну середню оцінку, яку повинен мати маршрут самостійної подорожі автомобілем.

З результатів можна перейти до перегляду конкретного маршруту самостійної подорожі автомобілем (рис. 4.2).

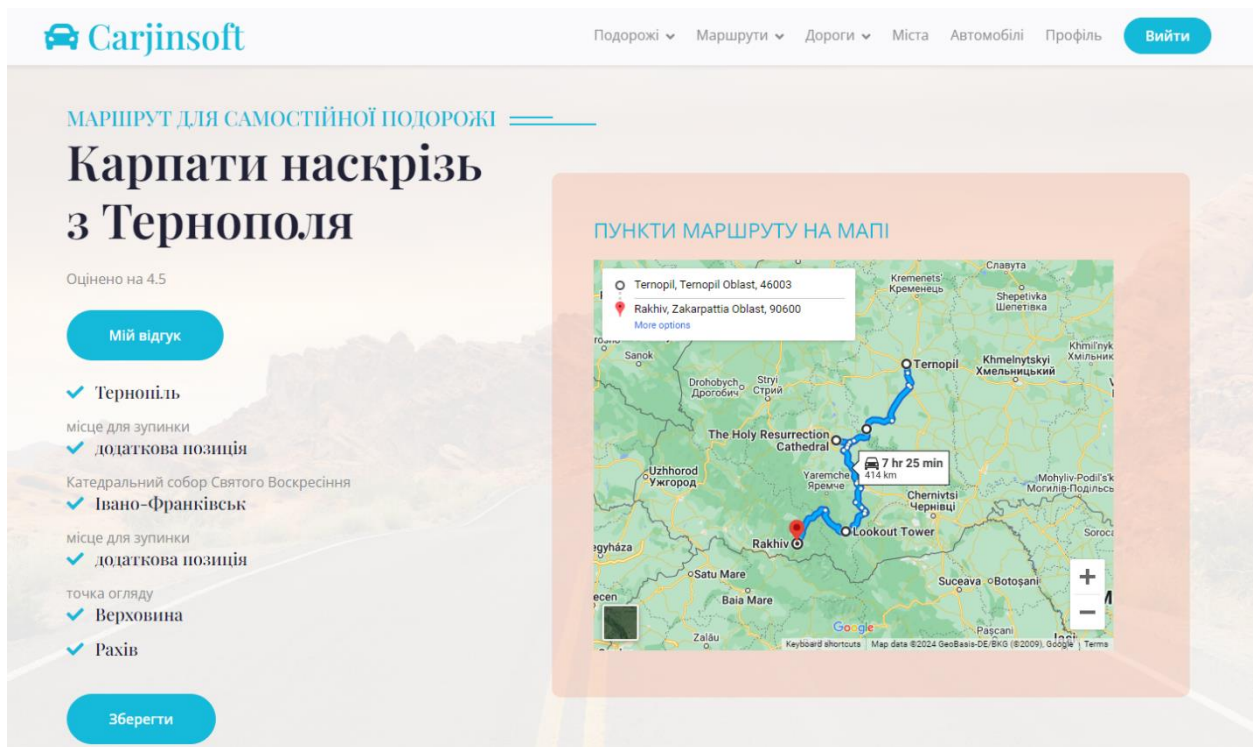


Рисунок 4.2 – Сторінка маршруту, який може використовуватися для самостійної подорожі автомобілем

Зі сторінки маршруту, який може використовуватися для самостійної подорожі автомобілем, можна перейти до створення власного відгуку, можна зберегти цей маршрут, щоб потім через меню відобразити всі такі маршрути. За маршрутом демонструється набір місць на мапі, які входять до нього (опис місця вище, а потім населений пункт або позначення «додаткова позиція» за відсутності прив'язаного населеного пункту). У правій частині на мапі за визначеними координатами показується маршрут подорожі. Будь-який маршрут може складатися з широковідомих доріг та з окремих місць на мапі. Далі на сторінці знаходяться відгуки про цей маршрут для самостійної подорожі автомобілем, а також можна перейти до сторінки зі всіма відгуками.

Розділ меню “Дороги” включає пункти для перегляду збережених доріг, відвіданих доріг та відгуків за дорогами користувача, який авторизувався. Відвідані дороги включають дороги з відгуків користувача, з явно визначених відвідувань доріг та з маршрутів описаних самостійних

подорожей автомобілем цього користувача, які включають цей маршрут (дорогу у його складі) до свого складу.

Відгуки авторизованого користувача про дороги (рис. 4.3) відображаються з можливістю перемикання між ними. За кожною дорогою міститься оцінка, текст, а також можна перейти до сторінки користувача, який створив відгук, натиснувши на фотографію, а також до сторінки дороги, натиснувши на фотографію дороги, що відображається внизу.

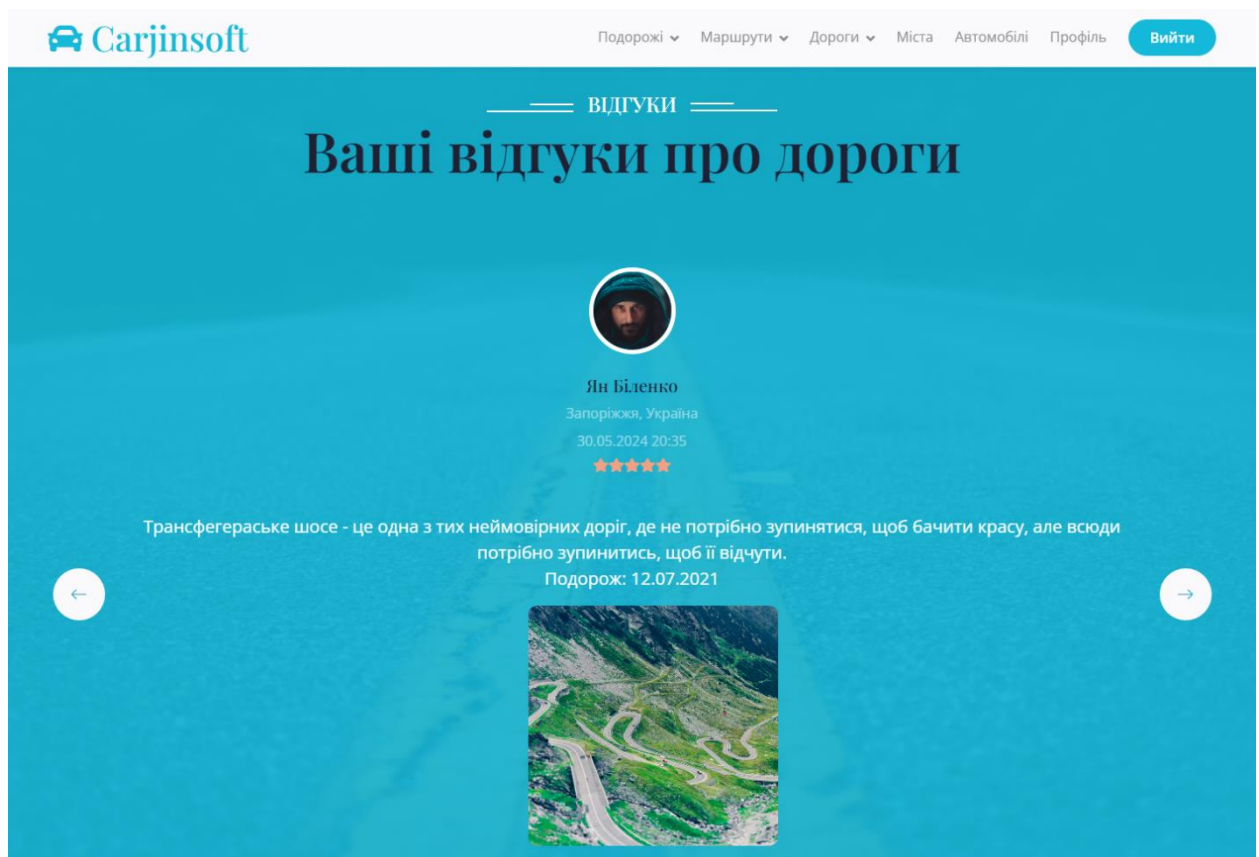


Рисунок 4.3 – Сторінка відгуків про широковідомі дороги авторизованого користувача

Розділ меню “Міста” містить пошук населених пунктів, де параметри пошуку включають:

- назву населеного пункту;
- країну, де має знаходитися населений пункт.

Розділ меню “Автомобілі” дозволяє переглядати всі автомобілі, які для себе додав авторизований користувач, а також вносити нові, якщо це потрібно.

Розділ меню “Профіль” надає можливість переглядати власні дані користувача, редагувати їх.

4.4 Висновки за розділом 4

Перевірка всіх функцій була здійснена і успішно завершена. Програма здатна забезпечувати підтримку самостійних подорожей автомобілем, дозволяючи надавати інформацію про досвід інших користувачів, шукати нові варіанти для подорожей, ділитися власним досвідом.

ВИСНОВКИ

Виконана робота реалізує розробку програмного забезпечення створення відгуків про самостійні подорожі автомобілем для обміну інформацією стосовно можливостей відвідування різних місць автомобілем.

Виконано вибір технологій розробки, куди входять мова Python з вебфреймворком Django, визначення сценаріїв роботи з програмою користувача як основних, так і альтернативних, визначення структури БД.

Розроблена програма реалізує ряд функцій, що включають формування маршруту, який може використовуватися для самостійної подорожі автомобілем, перегляд запису широковідомої дороги, перегляд всіх відгуків за широковідомою дорогою та за маршрутом самостійної подорожі автомобілем, перегляд всіх подорожей за широковідомою дорогою та за маршрутом самостійної подорожі автомобілем, перегляд маршруту самостійної подорожі автомобілем, додавання інформації про використовуваний автомобіль користувачем для самостійних подорожей, перегляд даних користувача, реалізацію опису виконаної самостійної подорожі автомобілем та перегляд цього опису, перегляд всіх самостійних подорожей автомобілем користувача, створення відгуку про маршрут самостійної подорожі або про широковідому дорогу, збереження широковідомої дороги та маршруту самостійної подорожі, перегляд збережених доріг, маршрутів самостійних подорожей, визначення відвідування дороги, перегляд всіх відвіданих користувачем доріг, перегляд відгуків користувача про маршрути самостійної подорожі, про дороги, пошук маршрутів самостійних подорожей автомобілем, виконаних подорожей, населених пунктів, перегляд запису населеного пункту.

Перевірка всіх функцій була здійснена і успішно завершена. Програма здатна забезпечувати підтримку самостійних подорожей автомобілем, дозволяючи надавати інформацію про досвід інших користувачів, шукати нові варіанти для подорожей, ділитися власним досвідом.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Furkot. My Trips [Electronic resource]. – Access mode : <https://trips.furkot.com/ui>.
2. Lifejourney4two [Electronic resource]. – Access mode : <https://lifejourney4two.com/>.
3. Блог Олени та Євгенія Пузатків [Електрон. ресурс]. – Режим доступу : <https://ua.elenapuzatko.com/>.
4. Alder, E. Road Trip [Electronic resource] : Camping out in the 2024 Land Rover Defender Outbound / E. Alder. – Access mode : <https://driving.ca/reviews/road-trip/road-trip-camping-out-in-the-2024-land-rover-defender-outbound>.
5. Django Tutorial in Visual Studio Code [Electronic resource]. – Access mode : <https://code.visualstudio.com/docs/python/tutorial-django>.
6. CherryPy – A Minimalist Python Web Framework [Electronic resource]. – Access mode : <https://docs.cherrypy.dev/en/latest/>.
7. GitHub – cherrypy/cherrypy [Electronic resource] : CherryPy is a pythonic, object-oriented HTTP framework. – Access mode : <https://github.com/cherrypy/cherrypy>.
8. Terapia – Free Bootstrap 5 HTML5 Fitness Website Template [Electronic resource]. – Access mode : <https://themewagon.com/themes/terapia/>.

ДОДАТОК А
Технічне завдання

Вступ

Подорожі можуть відбуватися в різних варіантах. Одним з таких варіантів є самостійні подорожі автомобілем. Особливістю цього типу подорожей є те, що самі подорожуючі формують маршрут подорожі, а тому досвід інших людей, які вже відвідували ці місця автомобілем, є корисним. Тому потрібно створити програму, яка дозволить для самостійних подорожей визначати маршрути, ділитися інформацією про виконані подорожі на основі цих маршрутів, залишати відгуки про використані для таких подорожей автомобілем маршрути та зручно шукати потрібну інформацію серед всієї відповідної накопиченої.

A.1 Підстави для розробки

Дипломна кваліфікаційна робота виконувалась у відповідності та на підставі Наказу № 168 від 24 квітня 2024 р. за Національним університетом «Запорізька політехніка», яким визначено тему роботи – «Програмне забезпечення створення відгуків про самостійні подорожі автомобілем».

A.2 Призначення розробки

Призначення програми в тому, щоб підтримувати самостійні подорожі автомобілем, дозволяючи надавати інформацію про досвід інших користувачів, шукати нові варіанти для подорожей, ділитися власним досвідом.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функціональні характеристики програми створення відгуків про

самостійні подорожі автомобілем повинні включати:

- формування маршруту, який може використовуватися для самостійної подорожі автомобілем;
- перегляд запису широковідомої дороги;
- перегляд всіх відгуків за широковідомою дорогою;
- перегляд всіх подорожей за широковідомою дорогою;
- перегляд маршруту самостійної подорожі автомобілем;
- перегляд всіх відгуків за маршрутом самостійної подорожі автомобілем;
- перегляд всіх подорожей за маршрутом самостійної подорожі автомобілем;
- реєстрація в програмі або авторизація;
- додавання інформації про використовуваний автомобіль користувачем для самостійних подорожей;
- перегляд даних користувача;
- редагування інформації про використовуваний автомобіль користувачем для самостійних подорожей;
- редагування даних користувача;
- опис виконаної самостійної подорожі автомобілем;
- редагування опису виконаної самостійної подорожі автомобілем;
- перегляд виконаної самостійної подорожі автомобілем;
- перегляд всіх самостійних подорожей автомобілем користувача;
- створення відгуку про маршрут самостійної подорожі;
- створення відгуку про широковідому дорогу;
- збереження широковідомої дороги;
- збереження маршруту самостійної подорожі;
- перегляд збережених доріг;
- перегляд збережених маршрутів самостійних подорожей;
- вилучення збереженої широковідомої дороги;
- вилучення збереженого маршруту самостійної подорожі;

- визначення відвідування дороги;
- перегляд всіх відвіданих користувачем доріг;
- перегляд відгуків користувача про маршрути самостійної подорожі;
- перегляд відгуків користувача про дороги;
- пошук маршрутів самостійних подорожей автомобілем;
- пошук виконаних самостійних подорожей автомобілем;
- пошук населених пунктів;
- перегляд запису населеного пункту.

A.3.2 Умови експлуатації

Обов'язковою умовою експлуатації програми створення відгуків про самостійні подорожі автомобілем є наявність адміністратора програми або користувача, здатного виконувати такі функції. В обов'язки адміністратора входить наповнення даними програми зокрема стосовно населених пунктів, широковідомих доріг. Також адміністратор повинен виконати розгортання додатку на сервері.

A.3.3 Вимоги до складу та параметрів технічних засобів

Технічне забезпечення сервера повинно включати мінімально:

- процесор Intel Xeon E5440 (частота 2,83 ГГц);
- жорсткий диск – 100 Гб (наявність фотографій може значно збільшувати вимоги до сервера, тому це потребує контролю та можливо введення обмежень на обсяг фотографій, які користувач може завантажувати, інакше необхідно розширювати потрібний простір);
- оперативна пам'ять – 4 Гб.

A.4 Порядок контролю та приймання

Починаючи з дати видачі завдання – 15 квітня, протягом наступних 8 тижнів має бути забезпечено виконання роботи за календарним планом, що містить в завданні на роботу. Календарний план визначає послідовність етапів виконання роботи та за кожним таким етапом – строк виконання і основний очікуваний результат. Після забезпечення цих очікуваних результатів формується пояснювальна записка до дипломної кваліфікаційної роботи. Остаточним кроком приймання роботи є її публічний захист.

ДОДАТОК Б
Текст програми

```

from .models import Road
from .models import Roadphoto
from .models import Drivetrackroad
from .models import Carjourney
from .models import Driveruser
from .models import Roadreview
from django.shortcuts import render
from .models import Roadposition
from .models import Drivetrack

def road_view(request, road_num):
    road_view_context = {}
    try:
        road_data = Road.objects.get(id = road_num)
    except Road.DoesNotExist:
        return render(request, "carjinsoft/error.html",
{"text": "Така дорога відсутня, введено некоректний
ідентифікатор."})
    road_view_context["road"] = road_data
    road_photos = Roadphoto.objects.filter(road =
road_data)
    road_view_context["photos"] = road_photos
    road_positions =
Roadposition.objects.select_related("drivecountry")
    road_positions = road_positions.filter(road =
road_data)
    road_view_context["positions"] = road_positions
    road_tracks =
Drivetrackroad.objects.select_related("drivetrack")
    road_tracks = road_tracks.filter(road = road_data)
    road_tracks = road_tracks.values_list("drivetrack")
    road_tracks = Drivetrack.objects.filter(id__in =
road_tracks)
    road_tracks = road_tracks.order_by("-
track_review")[:10]
    road_view_context["tracks"] = road_tracks
    road_journeys =
Carjourney.objects.filter(drivetracks__in = road_tracks)
    road_journeys = road_journeys.order_by("-
journey_end")[:10]
    road_view_context["journeys"] = road_journeys
    roadsave = False
    roadvisit = False
    roadreview = False

```

```

        if request.user.is_active:
            driver_user =
Driveruser.objects.select_related("auth_user")
            driver_user = driver_user.prefetch_related("roads")
            driver_user =
driver_user.prefetch_related("drivetracks")
            driver_user = driver_user.filter(auth_user =
request.user)
            driver_user = driver_user[0]
            if road_data in driver_user.roads:
                roadsave = True
                review = Roadreview.objects.filter(driveruser =
driver_user, road = road_data)
                if review:
                    roadreview = True
                    visit = Roadvisit.objects.filter(driveruser =
driver_user, road = road_data)
                    if visit:
                        roadvisit = True
                    for road_track in road_tracks:
                        if road_track in driver_user.drivetracks:
                            road_track.saved = True
                road_view_context["save"] = roadsave
                road_view_context["review"] = roadreview
                road_reviews =
Roadreview.objects.select_related("road")
                road_reviews =
road_reviews.select_related("driveruser")
                road_reviews = road_reviews.filter(road = road_data)
                road_reviews = road_reviews.order_by("-
visit_date")[:10]
                road_view_context["reviews"] = road_reviews
                return render(request, "carjinsoft/road.html",
road_view_context)

def all_road_reviews_view(request, road_num):
    road_view_context = {}
    try:
        road_data = Road.objects.get(id = road_num)
    except Road.DoesNotExist:
        return render(request, "carjinsoft/error.html",
{"text": "Така дорога відсутня, введено некоректний
ідентифікатор."})
    road_view_context["road"] = road_data

```

```

        road_photos = Roadphoto.objects.filter(road =
road_data)
        road_view_context["photos"] = road_photos[:3]
        road_reviews =
Roadreview.objects.select_related("road")
        road_reviews =
road_reviews.select_related("driveruser")
        road_reviews = road_reviews.filter(road = road_data)
        road_reviews = road_reviews.order_by("-visit_date")
        road_view_context["reviews"] = road_reviews
        roadsave = False
        if request.user.is_active:
            driver_user =
Driveruser.objects.select_related("auth_user")
            driver_user = driver_user.prefetch_related("roads")
            driver_user =
driver_user.prefetch_related("drivetracks")
            driver_user = driver_user.filter(auth_user =
request.user)
            driver_user = driver_user[0]
            if road_data in driver_user.roads:
                roadsave = True
        road_view_context["save"] = roadsave
        return render(request, "carjinsoft/roadreviews.html",
road_view_context)

def all_road_journeys_view(request, road_num):
    road_view_context = {}
    try:
        road_data = Road.objects.get(id = road_num)
    except Road.DoesNotExist:
        return render(request, "carjinsoft/error.html",
{"text": "Така дорога відсутня, введено некоректний
ідентифікатор."})
    road_view_context["road"] = road_data
    road_photos = Roadphoto.objects.filter(road =
road_data)
    road_view_context["photos"] = road_photos[:3]
    road_tracks =
Drivetrackroad.objects.select_related("drivetrack")
    road_tracks = road_tracks.filter(road = road_data)
    road_tracks = road_tracks.values_list("drivetrack")
    road_tracks = Drivetrack.objects.filter(id__in =
road_tracks)

```

```

        road_tracks = road_tracks.order_by("-track_review")
        road_view_context["tracks"] = road_tracks
        road_journeys =
Carjourney.objects.filter(drivetracks__in = road_tracks)
        road_journeys = road_journeys.order_by("-journey_end")
        road_view_context["journeys"] = road_journeys
        roadsave = False
        if request.user.is_active:
            driver_user =
Driveruser.objects.select_related("auth_user")
            driver_user = driver_user.prefetch_related("roads")
            driver_user =
driver_user.prefetch_related("drivetracks")
            driver_user = driver_user.filter(auth_user =
request.user)
            driver_user = driver_user[0]
            if road_data in driver_user.roads:
                roadsave = True
            road_view_context["save"] = roadsave
            return render(request, "carjinsoft/roadjourneys.html",
road_view_context)

from .models import Drivetrackposition

def drivetrack_view(request, drivetrack_num):
    drivetrack_view_context = {}
    try:
        drivetrack_data = Drivetrack.objects.get(id =
drivetrack_num)
    except Drivetrack.DoesNotExist:
        return render(request, "carjinsoft/error.html",
{"text": "Такий маршрут самостійної подорожі відсутній, введено
некоректний ідентифікатор."})
    drivetrack_view_context["drivetrack"] = drivetrack_data
    drivetrack_positions =
Drivetrackposition.objects.select_related("drivetrack")
    drivetrack_positions =
Drivetrackposition.objects.select_related("driveposition")
    drivetrack_positions =
Drivetrackposition.objects.select_related("driveposition__drives
ettlement")
    drivetrack_positions =
Drivetrackposition.objects.select_related("driveposition__drives
ettlement__drivecountry")

```

```

        drivetrack_positions =
drivetrack_positions.filter(drivetrack = drivetrack_data)
        drivetrack_positions =
drivetrack_positions.order_by("journey_number")
        road_view_context["positions"] = drivetrack_positions
        drivetrack_roads =
Drivetrackroad.objects.select_related("road")
        drivetrack_roads = drivetrack_roads.filter(drivetrack =
drivetrack_data)
        drivetrack_roads =
drivetrack_roads.order_by("journey_number")
        drivetrack_view_context["roads"] = drivetrack_roads
        drivetrack_reviews =
Drivetrackreview.objects.select_related("drivetrack")
        drivetrack_reviews =
drivetrack_reviews.select_related("driveruser")
        drivetrack_reviews =
drivetrack_reviews.filter(drivetrack = drivetrack_data)
        drivetrack_reviews = drivetrack_reviews.order_by("-
reviewed_time")[:10]
        drivetrack_view_context["reviews"] = drivetrack_reviews
        drivetrack_journeys =
Carjourney.objects.select_related("drivercar")
        drivetrack_journeys =
drivetrack_journeys.select_related("drivercar__driveruser")
        drivetrack_journeys =
drivetrack_journeys.prefetch_related("drivetracks")
        drivetrack_journeys =
drivetrack_journeys.filter(drivetracks__in = [drivetrack_data])
        drivetrack_journeys = drivetrack_journeys.order_by("-
journey_end")
        drivetrack_view_context["journeys"] =
drivetrack_journeys[:10]
        drivetracksave = False
        if request.user.is_active:
            driver_user =
Driveruser.objects.select_related("auth_user")
            driver_user =
driver_user.prefetch_related("drivetracks")
            driver_user = driver_user.filter(auth_user =
request.user)
            driver_user = driver_user[0]
            if drivetrack_data in driver_user.drivetracks:
                drivetracksave = True

```

```

        drivetrack_view_context["save"] = drivetracksave
        return render(request, "carjinsoft/drivetrack.html",
drivetrack_view_context)

    def all_drivetrack_reviews_view(request, drivetrack_num):
        drivetrack_view_context = {}
        try:
            drivetrack_data = Drivetrack.objects.get(id =
drivetrack_num)
        except Drivetrack.DoesNotExist:
            return render(request, "carjinsoft/error.html",
{"text": "Такий маршрут самостійної подорожі відсутній, введено
некоректний ідентифікатор."})
        drivetrack_view_context["drivetrack"] = drivetrack_data
        drivetrack_reviews =
Drivetrackreview.objects.select_related("drivetrack")
        drivetrack_reviews =
drivetrack_reviews.select_related("driveruser")
        drivetrack_reviews =
drivetrack_reviews.filter(drivetrack = drivetrack_data)
        drivetrack_reviews = drivetrack_reviews.order_by("-
reviewed_time")
        drivetrack_view_context["reviews"] = drivetrack_reviews
        drivetracksave = False
        if request.user.is_active:
            driver_user =
Driveruser.objects.select_related("auth_user")
            driver_user =
driver_user.prefetch_related("drivetracks")
            driver_user = driver_user.filter(auth_user =
request.user)
            driver_user = driver_user[0]
            if drivetrack_data in driver_user.drivetracks:
                drivetracksave = True
        drivetrack_view_context["save"] = drivetracksave
        return render(request,
"carjinsoft/drivetrackreviews.html", drivetrack_view_context)

    def all_drivetrack_journeys_view(request, drivetrack_num):
        drivetrack_view_context = {}
        try:
            drivetrack_data = Drivetrack.objects.get(id =
drivetrack_num)
        except Drivetrack.DoesNotExist:

```

```

        return render(request, "carjinsoft/error.html",
{"text": "Такий маршрут самостійної подорожі відсутній, введено некоректний ідентифікатор."})
        drivetrack_view_context["drivetrack"] = drivetrack_data
        drivetrack_journeys =
Carjourney.objects.select_related("drivercar")
        drivetrack_journeys =
drivetrack_journeys.select_related("drivercar__driveruser")
        drivetrack_journeys =
drivetrack_journeys.prefetch_related("drivetracks")
        drivetrack_journeys =
drivetrack_journeys.filter(drivetracks__in = [drivetrack_data])
        drivetrack_journeys = drivetrack_journeys.order_by("-
journey_end")
        drivetrack_view_context["journeys"] =
drivetrack_journeys
        drivetracksave = False
        if request.user.is_active:
            driver_user =
Driveruser.objects.select_related("auth_user")
            driver_user =
driver_user.prefetch_related("drivetracks")
            driver_user = driver_user.filter(auth_user =
request.user)
            driver_user = driver_user[0]
            if drivetrack_data in driver_user.drivetracks:
                drivetracksave = True
        drivetrack_view_context["save"] = drivetracksave
        return render(request,
"carjinsoft/drivetrackjourneys.html", drivetrack_view_context)

def all_saved_roads_view(request):
    road_view_context = {}
    if request.user.is_active:
        driver_user =
Driveruser.objects.select_related("auth_user")
        driver_user = driver_user.prefetch_related("roads")
        driver_user = driver_user.filter(auth_user =
request.user)
        driver_user = driver_user[0]
        roads = driver_user.roads
        for driver_road in roads:
            driver_road.photo =
Roadphoto.objects.filter(road = driver_road)[0]

```

```

        driver_road.reviews =
Roadreview.objects.filter(road = driver_road).count()
        visit = Roadvisit.objects.filter(driveruser =
driver_user, road = driver_road)
        if visit:
            driver_road.roadvisit = True
        else:
            driver_road.roadvisit = False
    else:
        return render(request, "carjinsoft/error.html",
{"text": "Збереження доступні тільки після авторизації."})
        road_view_context["roads"] = roads
        road_view_context["driver"] = driver_user
        return render(request, "carjinsoft/savedroads.html",
road_view_context)

def all_saved_drivetracks_view(request):
    drivetrack_view_context = {}
    if request.user.is_active:
        driver_user =
Driveruser.objects.select_related("auth_user")
        driver_user =
driver_user.prefetch_related("drivetracks")
        driver_user = driver_user.filter(auth_user =
request.user)
        driver_user = driver_user[0]
        drivetracks = driver_user.drivetracks
        for drivetrack in drivetracks:
            drivetrack.reviews =
Drivetrackreview.objects.filter(drivetrack = drivetrack).count()
        else:
            return render(request, "carjinsoft/error.html",
{"text": "Збереження доступні тільки після авторизації."})
            drivetrack_view_context["drivetracks"] = drivetracks
            drivetrack_view_context["driver"] = driver_user
            return render(request,
"carjinsoft/saveddrivetracks.html", drivetrack_view_context)

from .models import Carjourneyphoto

def driveruser_journeys_view(request, driver_user_name =
None):
    carjourney_view_context = {}
    if not driver_user_name and request.user.is_active:

```

```

        driver_user =
Driveruser.objects.select_related("auth_user")
        driver_user = driver_user.prefetch_related("roads")
        driver_user = driver_user.filter(auth_user =
request.user)
        driver_user = driver_user[0]
        carjourneys =
Carjourney.objects.select_related("drivercar")
        carjourneys =
carjourneys.select_related("drivercar__driveruser")
        carjourneys =
carjourneys.select_related("drivercar__carmanufmodel")
        carjourneys =
carjourneys.select_related("drivercar__carmanufmodel__carmanufac
turer")
        carjourneys =
carjourneys.filter(drivercar__driveruser = driver_user)
        carjourneys = carjourneys.order_by("-journey_end")
        for carjourney in carjourneys:
            carjourney.photos =
Carjourneyphoto.objects.filter(carjourney = carjourney)
            carjourney_view_context["driveruser"] = driver_user
            carjourney_view_context["journeys"] = carjourneys
            return render(request,
"carjinsoft/driverjourneys.html", carjourney_view_context)
        elif driver_user_name:
            driver_user =
Driveruser.objects.select_related("auth_user")
            driver_user = driver_user.prefetch_related("roads")
            driver_user =
driver_user.filter(auth_user__username = driver_user_name)
            driver_user = driver_user[0]
            carjourneys =
Carjourney.objects.select_related("drivercar")
            carjourneys =
carjourneys.select_related("drivercar__driveruser")
            carjourneys =
carjourneys.select_related("drivercar__carmanufmodel")
            carjourneys =
carjourneys.select_related("drivercar__carmanufmodel__carmanufac
turer")
            carjourneys =
carjourneys.filter(drivercar__driveruser = driver_user)
            carjourneys = carjourneys.order_by("-journey_end")

```

```

        for carjourney in carjourneys:
            carjourney.photos =
Carjourneyphoto.objects.filter(carjourney = carjourney)
            carjourney_view_context["driveruser"] = driver_user
            carjourney_view_context["journeys"] = carjourneys
            return render(request,
"carjinsoft/driverjourneys.html", carjourney_view_context)
        else:
            return render(request, "carjinsoft/error.html",
{"text": "Потрібно або авторизуватися, або задати
користувача."})

def driveruser_visited_roads_view(request, driver_user_name
= None):
    road_view_context = {}
    if not driver_user_name and request.user.is_active:
        driver_user =
Driveruser.objects.select_related("auth_user")
        driver_user = driver_user.prefetch_related("roads")
        driver_user = driver_user.filter(auth_user =
request.user)
        driver_user = driver_user[0]
        roads_from_reviews =
Roadreview.objects.filter(driveruser = driver_user)
        roads_from_reviews = roads_from_reviews.order_by("-
visit_date")
        roads_from_visits =
Roadvisit.objects.filter(driveruser = driver_user)
        roads_from_visits = roads_from_visits.order_by("-
visit_date")
        roads_from_journeys =
Carjourney.objects.prefetch_related("drivetracks")
        roads_from_journeys =
roads_from_journeys.filter(drivercar__driveruser = driver_user)
        roads_from_journeys =
roads_from_journeys.values_list("drivetracks")
        drivetrackroads =
Drivetrackroad.objects.select_related("drivetrack")
        drivetrackroads =
drivetrackroads.select_related("road")
        roads_from_journeys =
drivetrackroads.filter(drivetrack__in = roads_from_journeys)
        roads_from_journeys =
roads_from_journeys.values_list("road")

```

```

        all_roads = []
        for road_from_review in roads_from_reviews:
            road_from_review.photo =
Roadphoto.objects.filter(road = road_from_review)[0]
            road_from_review.reviews =
Roadreview.objects.filter(road = road_from_review).count()
            visit = Roadvisit.objects.filter(driveruser =
driver_user, road = road_from_review)
            if visit:
                road_from_review.roadvisit = True
            else:
                road_from_review.roadvisit = False
                all_roads.append(road_from_review)
        for roads_from_visit in roads_from_visits:
            roads_from_visit.photo =
Roadphoto.objects.filter(road = roads_from_visit)[0]
            roads_from_visit.reviews =
Roadreview.objects.filter(road = roads_from_visit).count()
            visit = Roadvisit.objects.filter(driveruser =
driver_user, road = roads_from_visit)
            if visit:
                roads_from_visit.roadvisit = True
            else:
                roads_from_visit.roadvisit = False
                all_roads.append(roads_from_visit)
        for roads_from_journey in roads_from_journeys:
            roads_from_journey.photo =
Roadphoto.objects.filter(road = roads_from_journey)[0]
            roads_from_journey.reviews =
Roadreview.objects.filter(road = roads_from_journey).count()
            visit = Roadvisit.objects.filter(driveruser =
driver_user, road = roads_from_journey)
            if visit:
                roads_from_journey.roadvisit = True
            else:
                roads_from_journey.roadvisit = False
                all_roads.append(roads_from_journey)
        road_view_context["driveruser"] = driver_user
        road_view_context["roads"] = all_roads
        return render(request,
"carjinsoft/visitedroads.html", road_view_context)
    elif driver_user_name:
        driver_user =
Driveruser.objects.select_related("auth_user")

```

```

        driver_user = driver_user.prefetch_related("roads")
        driver_user =
driver_user.filter(auth_user__username = driver_user_name)
        driver_user = driver_user[0]
        roads_from_reviews =
Roadreview.objects.filter(driveruser = driver_user)
        roads_from_reviews = roads_from_reviews.order_by("-
visit_date")
        roads_from_visits =
Roadvisit.objects.filter(driveruser = driver_user)
        roads_from_visits = roads_from_visits.order_by("-
visit_date")
        roads_from_journeys =
Carjourney.objects.prefetch_related("drivetracks")
        roads_from_journeys =
roads_from_journeys.filter(drivercar__driveruser = driver_user)
        roads_from_journeys =
roads_from_journeys.values_list("drivetracks")
        drivetrackroads =
Drivetrackroad.objects.select_related("drivetrack")
        drivetrackroads =
drivetrackroads.select_related("road")
        roads_from_journeys =
drivetrackroads.filter(drivetrack__in = roads_from_journeys)
        roads_from_journeys =
roads_from_journeys.values_list("road")
        all_roads = []
        for road_from_review in roads_from_reviews:
            road_from_review.photo =
Roadphoto.objects.filter(road = road_from_review)[0]
            road_from_review.reviews =
Roadreview.objects.filter(road = road_from_review).count()
            visit = Roadvisit.objects.filter(driveruser =
driver_user, road = road_from_review)
            if visit:
                road_from_review.roadvisit = True
            else:
                road_from_review.roadvisit = False
            all_roads.append(road_from_review)
        for roads_from_visit in roads_from_visits:
            roads_from_visit.photo =
Roadphoto.objects.filter(road = roads_from_visit)[0]
            roads_from_visit.reviews =
Roadreview.objects.filter(road = roads_from_visit).count()

```

```

        visit = Roadvisit.objects.filter(driveruser =
driver_user, road = roads_from_visit)
        if visit:
            roads_from_visit.roadvisit = True
        else:
            roads_from_visit.roadvisit = False
            all_roads.append(roads_from_visit)
        for roads_from_journey in roads_from_journeys:
            roads_from_journey.photo =
Roadphoto.objects.filter(road = roads_from_journey)[0]
            roads_from_journey.reviews =
Roadreview.objects.filter(road = roads_from_journey).count()
            visit = Roadvisit.objects.filter(driveruser =
driver_user, road = roads_from_journey)
            if visit:
                roads_from_journey.roadvisit = True
            else:
                roads_from_journey.roadvisit = False
                all_roads.append(roads_from_journey)
            road_view_context["driveruser"] = driver_user
            road_view_context["roads"] = all_roads
            return render(request,
"carjinsoft/visitedroads.html", road_view_context)
        else:
            return render(request, "carjinsoft/error.html",
{"text": "Потрібно або авторизуватися, або задати
користувача."})

```

```

def driveuser_road_reviews_view(request, driver_user_name =
None):
    road_view_context = {}
    if not driver_user_name and request.user.is_active:
        driver_user =
Driveruser.objects.select_related("auth_user")
        driver_user = driver_user.prefetch_related("roads")
        driver_user = driver_user.filter(auth_user =
request.user)
        driver_user = driver_user[0]
        road_reviews =
Roadreview.objects.select_related("road")
        road_reviews =
road_reviews.select_related("driveruser")
        road_reviews = road_reviews.filter(driveruser =
driver_user)

```

```

        road_reviews = road_reviews.order_by("-visit_date")
        for road_review in road_reviews:
            if road_review.road in driver_user.roads:
                road_review.save = True
            else:
                road_review.save = False
        road_view_context["reviews"] = road_reviews
        return render(request,
"carjinsoft/duroadreviews.html", road_view_context)
    elif driver_user_name:
        driver_user =
Driveruser.objects.select_related("auth_user")
        driver_user = driver_user.prefetch_related("roads")
        driver_user =
driver_user.filter(auth_user__username = driver_user_name)
        driver_user = driver_user[0]
        road_reviews =
Roadreview.objects.select_related("road")
        road_reviews =
road_reviews.select_related("driveruser")
        road_reviews = road_reviews.filter(driveruser =
driver_user)
        road_reviews = road_reviews.order_by("-visit_date")
        for road_review in road_reviews:
            if road_review.road in driver_user.roads:
                road_review.save = True
            else:
                road_review.save = False
        road_view_context["reviews"] = road_reviews
        return render(request,
"carjinsoft/duroadreviews.html", road_view_context)
    else:
        return render(request, "carjinsoft/error.html",
{"text": "Потрібно або авторизуватися, або задати
користувача."})

def driveuser_drivetrack_reviews_view(request,
driver_user_name = None):
    drivetrack_view_context = {}
    if not driver_user_name and request.user.is_active:
        driver_user =
Driveruser.objects.select_related("auth_user")
        driver_user =
driver_user.prefetch_related("drivetracks")

```

```

        driver_user = driver_user.filter(auth_user =
request.user)
        driver_user = driver_user[0]
        drivetrack_reviews =
Drivetrackreview.objects.select_related("drivetrack")
        drivetrack_reviews =
road_reviews.select_related("driveruser")
        drivetrack_reviews =
drivetrack_reviews.filter(driveruser = driver_user)
        drivetrack_reviews =
drivetrack_reviews.order_by("reviewed_time")
        for drivetrack_review in drivetrack_reviews:
            if drivetrack_review.drivetrack in
driver_user.drivetracks:
                drivetrack_review.save = True
            else:
                drivetrack_review.save = False
        drivetrack_view_context["reviews"] =
drivetrack_reviews
        return render(request,
"carjinsoft/dudrivetrackreviews.html", drivetrack_view_context)
    elif driver_user_name:
        driver_user =
Driveruser.objects.select_related("auth_user")
        driver_user = driver_user.prefetch_related("roads")
        driver_user =
driver_user.filter(auth_user__username = driver_user_name)
        driver_user = driver_user[0]
        drivetrack_reviews =
Drivetrackreview.objects.select_related("drivetrack")
        drivetrack_reviews =
road_reviews.select_related("driveruser")
        drivetrack_reviews =
drivetrack_reviews.filter(driveruser = driver_user)
        drivetrack_reviews =
drivetrack_reviews.order_by("reviewed_time")
        for drivetrack_review in drivetrack_reviews:
            if drivetrack_review.drivetrack in
driver_user.drivetracks:
                drivetrack_review.save = True
            else:
                drivetrack_review.save = False
        drivetrack_view_context["reviews"] =
drivetrack_reviews

```

```

        return render(request,
"carjinsoft/dudrivetrackreviews.html", drivetrack_view_context)
        else:
            return render(request, "carjinsoft/error.html",
{"text": "Потрібно або авторизуватися, або задати
користувача."})

from .models import Drivesettlement

def drivesettlement_view(request, drivesettlement_num):
    drivesettlement_view_context = {}
    try:
        drivesettlement_data =
Drivesettlement.objects.get(id = drivesettlement_num)
    except Drivesettlement.DoesNotExist:
        return render(request, "carjinsoft/error.html",
{"text": "Такий населений пункт відсутній, введено некоректний
ідентифікатор."})
    drivesettlement_view_context["settlement"] =
drivesettlement_data
    drivetracks =
Drivetrackposition.objects.select_related("drivetrack")
    drivetracks =
drivetracks.select_related("driveposition")
    drivetracks =
drivetracks.select_related("driveposition__drivesettlement")
    drivetracks =
drivetracks.filter(driveposition__drivesettlement =
drivesettlement_data)
    drivetracks = drivetracks.order_by("-track_review")
    if request.user.is_active:
        driver_user =
Driveruser.objects.select_related("auth_user")
        driver_user =
driver_user.prefetch_related("drivetracks")
        driver_user = driver_user.filter(auth_user =
request.user)
        driver_user = driver_user[0]
        for drivetrack in drivetracks:
            drivetrack.reviews =
Drivetrackreview.objects.filter(drivetrack = drivetrack).count()
    drivesettlement_view_context["drivetracks"] =
drivetracks

```

```

        return render(request, "carjinsoft/settlement.html",
drivesettlement_view_context)

def carjourney_view(request, carjourney_num):
    carjourney_view_context = {}
    try:
        carjourney_data =
Carjourney.objects.prefetch_related("drivetracks")
        carjourney_data =
carjourney_data.select_related("drivercar")
        carjourney_data =
carjourney_data.select_related("drivercar__driveruser")
        carjourney_data =
carjourney_data.select_related("drivercar__auth_user")
        carjourney_data =
carjourney_data.select_related("drivercar__drivesettlement")
        carjourney_data = carjourney_data.get(id =
carjourney_num)
    except Carjourney.DoesNotExist:
        return render(request, "carjinsoft/error.html",
{"text": "Така самотійна подорож автомобілем відсутня, введено
некоректний ідентифікатор."})
        carjourney_data.photos =
Carjourneyphoto.objects.filter(carjourney = carjourney_data)
        drivetracks = carjourney_data.drivetracks
        for drivetrack in drivetracks:
            drivetrack_positions =
Drivetrackposition.objects.select_related("drivetrack")
            drivetrack_positions =
Drivetrackposition.objects.select_related("driveposition")
            drivetrack_positions =
Drivetrackposition.objects.select_related("driveposition__drives
ettlement")
            drivetrack_positions =
Drivetrackposition.objects.select_related("driveposition__drives
ettlement__drivecountry")
            drivetrack_positions =
drivetrack_positions.filter(drivetrack = drivetrack_data)
            drivetrack_positions =
drivetrack_positions.order_by("journey_number")
            drivetrack.positions = drivetrack_positions
            drivetrack_roads =
Drivetrackroad.objects.select_related("road")

```

```

        drivetrack_roads =
drivetrack_roads.filter(drivetrack = drivetrack)
        drivetrack_roads =
drivetrack_roads.order_by("journey_number")
        drivetrack.roads = drivetrack_roads
        carjourney_view_context["carjourney"] = carjourney_data
        if request.user.is_active:
            driver_user =
Driveruser.objects.select_related("auth_user")
            driver_user =
driver_user.prefetch_related("drivetacks")
            driver_user = driver_user.filter(auth_user =
request.user)
            driver_user = driver_user[0]
            for drivetrack in drivetracks:
                if drivetrack in driver_user.drivetacks:
                    drivetrack.save = True
                else:
                    drivetrack.save = False
            drivetrack_view_context["save"] = drivetracksave
            carjourney_view_context["drivetacks"] = drivetracks
            return render(request, "carjinsoft/carjourney.html",
carjourney_view_context)

```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Дипломна кваліфікаційна робота бакалавра на тему:

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СТВОРЕННЯ ВІДГУКІВ ПРО САМОСТІЙНІ ПОДОРОЖІ АВТОМОБІЛЕМ. SOFTWARE FOR REVIEWING INDEPENDENT CAR JOURNEYS

Виконав студент групи КНТ-110
Керівник доцент, к.т.н.

Ростислав КОВАЛЬОВ
Степан СКРУПСЬКИЙ

2024

Рисунок В.1 – Перший слайд

ОБ'ЄКТ, ПРЕДМЕТ, МЕТА РОБОТИ

- об'єкт роботи – процес самостійного подорожування автомобілем;
- предмет роботи – програмне забезпечення створення відгуків про самостійні подорожі автомобілем;
- мета роботи – розробити програмне забезпечення створення відгуків про самостійні подорожі автомобілем для обміну інформацією стосовно можливостей відвідування різних місць автомобілем.

2

Рисунок В.2 – Об'єкт, предмет, мета роботи

ЗАВДАННЯ РОБОТИ

- проаналізувати програми створення відгуків про самостійні подорожі автомобілем;
- визначити засоби проектування програми;
- виділити вимоги до програми створення відгуків про самостійні подорожі автомобілем;
- виконати реалізацію програми створення відгуків про самостійні подорожі автомобілем.

3

Рисунок В.3 – Завдання роботи

ПОРІВНЯННЯ РОЗРОБКИ ПРОГРАМИ СТВОРЕННЯ ВІДГУКІВ ПРО САМОСТІЙНІ ПОДРОЖІ АВТОМОБІЛЕМ ТА АНАЛОГІВ

Критерій	Furkot	Блоги	Розробка
Пошук маршрутів	Відсутній	Тільки загальний текстовий	+
Створення власних маршрутів користувачем	+	–	+
Визначення власного досвіду подорожей	–	+	+
Створення відгуків про наявні маршрути	–	–	+

4

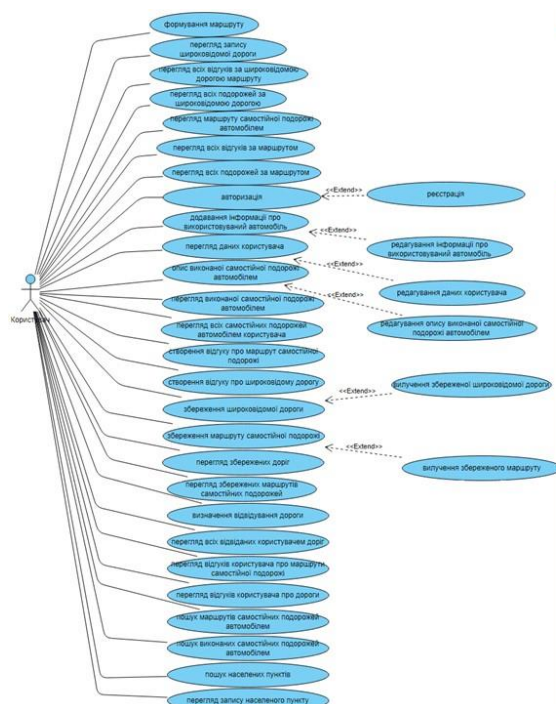
Рисунок В.4 – Порівняння розробки програми створення відгуків про самостійні подорожі автомобілем та аналогів

ВИБІР ФРЕЙМВОРКУ ДЛЯ РОЗРОБКИ ПРОГРАМИ СТВОРЕННЯ ВІДГУКІВ ПРО САМОСТІЙНІ ПОДОРОЖІ АВТОМОБІЛЕМ

Критерій	CherryPy	Django
Тип	Мініфреймворк	Мегафреймворк
Продуктивність	Висока	Прийнятна для використання в реальних додатках
Забезпечення підтримки баз даних	Потрібно обрати, встановити і використати фреймворк об'єктно-реляційного відображення	Усі необхідні засоби початково вбудовані
Адміністративна панель	Як такої вбудованої немає	Має вебінтерфейс, дозволяє змінювати, додавати, вилучати дані всіх моделей
Рухий для роботи з шаблонами	Потрібно обрати, встановити і використати	Вбудований власний (може бути змінений)
Безпека	Вбудованих інструментів немає	Великий набір вбудованих інструментів

5

Рисунок В.5 – Вибір фреймворку для розробки програми створення відгуків про самостійні подорожі автомобілем



ДІАГРАМА ПРЕЦЕДЕНТІВ ПРОГРАМИ СТВОРЕННЯ ВІДГУКІВ ПРО САМОСТІЙНІ ПОДОРОЖІ АВТОМОБІЛЕМ

6

Рисунок В.6 – Діаграма прецедентів програми створення відгуків про самостійні подорожі автомобілем

СХЕМА ФУНКЦІОНУВАННЯ ПРОГРАМИ СТВОРЕННЯ ВІДГУКІВ ПРО САМОСТІЙНІ ПОДОРОЖІ АВТОМОБІЛЕМ



Рисунок В.8 – Схема функціонування програми створення відгуків про
самостійні подорожі автомобілем

СТОРІНКА ОПИСУ ВИКОНАНОЇ САМОСТІЙНОЇ ПОДОРОЖІ АВТОМОБІЛЕМ

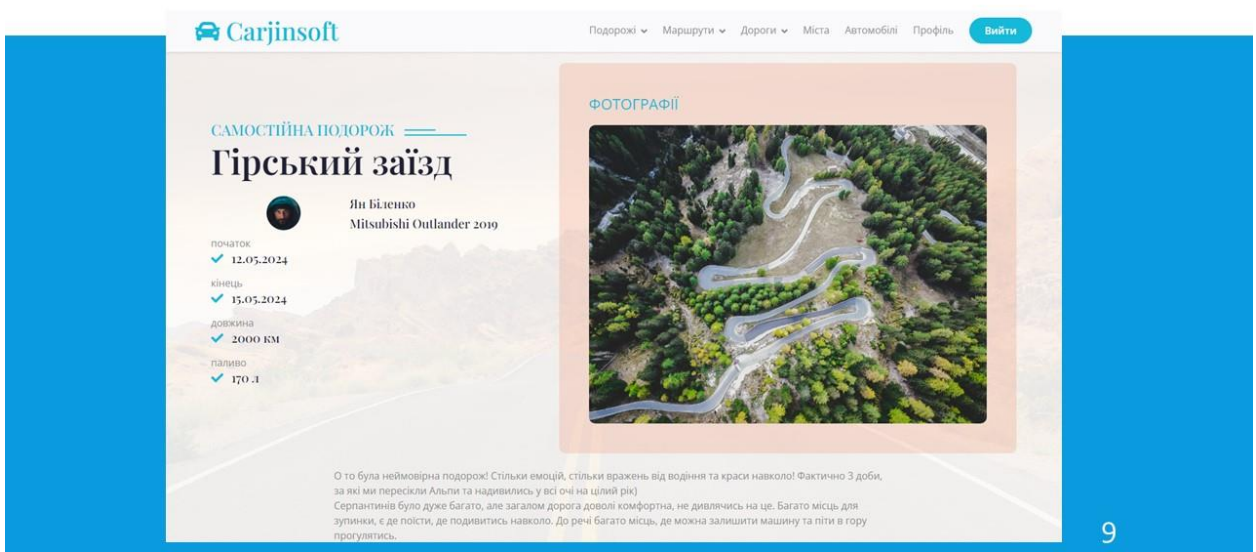


Рисунок В.9 – Сторінка опису виконаної самостійної подорожі автомобілем

СТОРІНКА МАРШРУТУ, ЯКИЙ МОЖЕ ВИКОРИСТОВУВАТИСЯ ДЛЯ САМОСТІЙНОЇ ПОДОРОЖІ АВТОМОБІЛЕМ

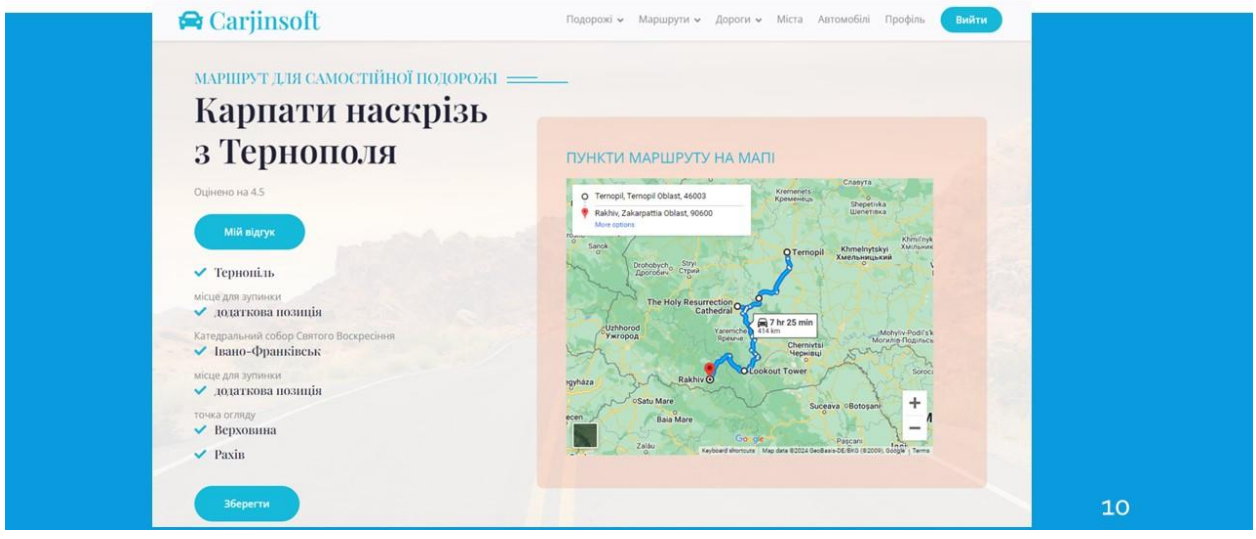


Рисунок В.10 – Сторінка маршруту, який може використовуватися для самостійної подорожі автомобілем

СТОРІНКА ВІДГУКІВ ПРО ШИРОКОВІДОМІ ДОРОГИ АВТОРИЗОВАНОГО КОРИСТУВАЧА

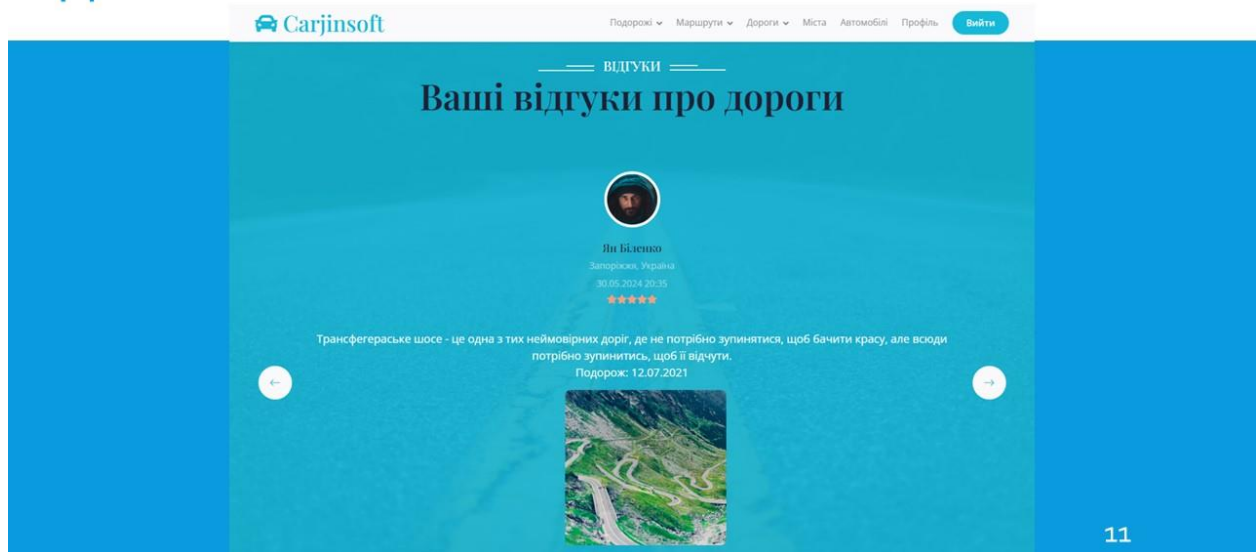


Рисунок В.11 – Сторінка відгуків про широковідомі дороги авторизованого користувача

ВИСНОВКИ

- виконана робота реалізує розробку програмного забезпечення створення відгуків про самостійні подорожі автомобілем для обміну інформацією стосовно можливостей відвідування різних місць автомобілем;
- виконано вибір технологій розробки, куди входять мова Python з вебфреймворком Django, визначення сценаріїв роботи з програмою користувача як основних, так і альтернативних, визначення структури бази даних;
- перевірка всіх функцій була здійснена і успішно завершена, програма здатна забезпечувати підтримку самостійних подорожей автомобілем, дозволяючи надавати інформацію про досвід інших користувачів, шукати нові варіанти для подорожей, ділитися власним досвідом.

12

Рисунок В.12 – Висновки